UNIVERSIDAD FRANCISCO DE VITORIA

Escuela Politécnica Superior

GRADO INGENIERÍA INFORMÁTICA

Proyecto Fin de Grado

Gestión de Reservas Online del Servicio de Deportes del Campus de la Universidad Francisco de Vitoria

Autor: **Ignacio López Chamorro**

Tutor: **Juan Pueyo Candil**

Convocatoria: **Julio 2015**

VISTO BUENO DEL TUTOR

Título: _____

Autor: _____

Tutor: _____

VISTO BUENO

VºBº Tutor del PFG:
Fdo.:

Lugar y fecha: _____

CALIFICACIÓN DEL PROYECTO FINAL DE GRADO

CUALITATIVA:	
NUMÉRICA:	

Conforme Presidente:	Conforme Secretario:	Conforme Vocal 1:
Fdo.:	Fdo.:	Fdo.:

Conforme Vocal 2:	Conforme Vocal 3:
Fdo.:	Fdo.:

Lugar y fecha: _____

Dedicatoria

Dedico este proyecto de fin grado en primer y destacado lugar a mis padres, Antonia y José Luis, les agradezco toda la dedicación incondicional en tantos aspectos de mi vida, y especialmente, el énfasis que pusieron en inculcarme los valores de esfuerzo, respeto, sacrificio y compromiso. A mi hermana Noemí, por haberme soportado en tantos momentos. Gracias a ellos hoy soy la persona que soy.

En segundo lugar, a Beatriz, que me acompaña en todo este camino y que me animó a iniciar esta etapa y me ha acompañado durante toda ella, gracias por su paciencia, por su voluntad y por su respeto continuado, por su comprensión y por todo lo que me ha dado. A nuestro hijo Gonzalo, para que aprenda que la familia y los valores son importantes, y es lo que le hará triunfar en la vida.

También quiero dedicar este proyecto a mi otra familia, que desde hace menos tiempo me viene apoyando en todo mi camino, Lucía, Miguel Ángel, Elsa, Montse y Lucía.

A mis compañeros en la Universidad, ha sido un camino duro y largo, pero con ellos ha sido mucho más llevadero, les agradezco todo este tiempo dedicado, todo el apoyo recibido y tantos buenos momentos compartidos.

Por último, querría mencionar a todos los profesores de la UFV, he tenido la suerte de enriquecer mi camino al encontrarme con estupendas personas, y en especial, a mi tutor, Juan, que me ha ayudado a culminar esta etapa y que ha formado parte de ese camino.

Resumen

Actualmente existen todo tipo de aplicaciones de gestión para distintos ámbitos pero llama la atención el déficit tecnológico asociado a la práctica del deporte no profesional, esto genera una serie de trabas como consulta de pistas deportivas libres, reserva de las mismas o incluso su pago. Estas situaciones redundan en una calidad del servicio inferior, obligando al usuario a tener que acudir personalmente o realizar la gestión vía telefónica de estos trámites, haciendo que el desarrollo del negocio dependa de horarios laborales del personal por no hablar por la falta de control y pérdida de datos de uso de las instalaciones. También la creación de estadísticas de uso y rendimiento para la toma de decisiones del equipo directivo es sumamente costosa con la gestión tradicional además de tardía ya que se carece de datos en tiempo real.

La realización de este proyecto está enfocada al análisis, diseño e implementación de una aplicación web en el que los usuarios podrán realizar la consulta y reserva de pistas deportivas desde su pc, móvil o tablet. Mientras que los administradores del Servicio de Deportes del Campus de la Universidad Francisco de Vitoria podrán gestionar esas reservas y la disponibilidad de pistas.

Palabras clave:

Aplicación web, central de reservas, JavaScript, Backbone.js, mysql, html5, css3

Abstract

There are all kinds of management applications for different areas but it is critical the lack of technical solutions focused on amateur sport activity, this generates a range of obstacles such as checking sport facilities availability, booking them or even the payment process. These situations result in lower quality of service, requiring the user to manage them in person or in best case by phone, all these things make the business development

depends on working hours and staff availability. It is important to highlight that this kind of tool will help a lot in terms of metrics, use of courts, sports data, or seasonal data.

The development of this project is focused on the analysis, design and implementation of a web application in which users can check availability and book sport facilities from their PC, mobile phone or tablet. While managers of the Sporting Area of Francisco de Vitoria University will manage those reserves and runway availability.

Keywords:

Web application, booking system, JavaScript, backbone.js, mysql, html5, css3

ÍNDICE

1. Introducción 1
   1.1. Resumen extendido 1
2. Descripción del problema 2
   2.1. Análisis de situación actual 2
   2.2. Objetivos del PFG 3
   2.3. Metodología 4
   2.4. Planificación del proyecto 5
   2.5. Restricciones 9
3. Detalle del proyecto 9
   3.1. Identificación de requisitos 9
   3.2. Definición de la arquitectura tecnológica 18
   3.3. Entorno de desarrollo 29
   3.4. Catálogo de requisitos 37
   3.5. Especificación de casos de uso 40
   3.6. Diagramas de actividad 55
   3.7. Diagramas de navegabilidad 67
   3.8. Definición de los servicios desarrollados 68
4. Resultados obtenidos 86
5. Conclusiones 87
6. Trabajos futuros 88
7. Bibliografía 90
8. Anexos 92
   8.1. Tutorial de instalación y uso 92

ÍNDICE DE FIGURAS

Figura 1: Plan de trabajo Marzo 7

Figura 2: Plan de trabajo Abril 7

Figura 3: Plan de trabajo Mayo 8

Figura 4: Plan de trabajo Junio-Julio 8

Figura 5: Definición arquitectura tecnológica 18

Figura 6: Modelo Entidad-Relación BBDD 20

Figura 7: Modelo control de versiones Git 32

Figura 8: Directorios principales de un proyecto en Git 33

Figura 9: Modelo relación roles de usuario 69

Figura 10: Pantalla de Registro 70

Figura 11: Pantalla Login 71

Figura 12: Pantalla Listado Deportes 75

Figura 13: Pantalla Selección de pista y Calendario 79

Figura 14: Pantalla listado Reservas con perfil usuario 81

Figura 15: Listado usuarios desde perfil administrador 84

Figura 16: Instalación de XAMPP 93

Figura 17: Instalación de XAMPP (continuación) 93

Figura 18: Panel de control XAMPP 94

Figura 19: Configuración archivo hosts 96

Figura 20: Configuración Rewrite en httpd.conf 98

Figura 21: Configuración httpd-vhosts.conf 99

ÍNDICE DE DIAGRAMAS

Diagrama 1: Procesos competencia de los administradores 10
Diagrama 2: Procesos competencia de los usuarios 10
Diagrama 3: E-R de la validación o verificación de datos 13
Diagrama 4: E-R de transacción en la base de datos 13
Diagrama 5: E-R para ver usuarios 14
Diagrama 6: E-R de creación de un deporte 14
Diagrama 7: E-R para borrar un deporte 14
Diagrama 8: E-R de creación de una nueva pista 15
Diagrama 9: E-R para ver las pistas 15
Diagrama 10: E-R de creación de reserva 15
Diagrama 11: E-R para ver reservas 16
Diagrama 12: E-R de cancelación de una reserva 16
Diagrama 13: E-R de creación de una nueva reserva 17
Diagrama 14: E-R para ver una reserva 17
Diagrama 15: E-R de modificación de datos personales 17
Diagrama 16: Casos de Uso 0 – Servicio de Deportes 41
Diagrama 17: Casos de uso 1 – Gestión de Deportes 45
Diagrama 18: Casos de uso 2 – Gestión de Pistas 48
Diagrama 19: Casos de uso 3 – Gestión de Usuarios 50
Diagrama 20: Casos de uso 4 – Gestión de Reservas 53
Diagrama 21: Diagrama de actividad para nuevo deporte 55
Diagrama 22: Diagrama de actividad para ver los deportes 56
Diagrama 23: Diagrama de actividad para modificar un deporte 57
Diagrama 24: Diagrama de actividad para eliminar un deporte 57

Diagrama 25: Diagrama de actividad para crear una nueva pista 58

Diagrama 26: Diagrama de actividad para ver las pistas.....59

Diagrama 27: Diagrama de actividad para borrar una pista 59

Diagrama 28: Diagrama de actividad para crear una nueva reserva 60

Diagrama 29: Diagrama de actividad para ver una reserva.....61

Diagrama 30: Diagrama de actividad para la cancelación de una reserva 62

Diagrama 31: Diagrama de actividad para registrarse un usuario.....63

Diagrama 32: Diagrama de actividad para ver usuarios.....63

Diagrama 33: Diagrama de actividad para modificar datos personales 64

Diagrama 34: Diagrama de actividad para borrar un usuario.....65

Diagrama 35: Diagrama de actividad de inicio de sesión 66

Diagrama 36: Diagrama de navegabilidad del administrador 67

Diagrama 37: Diagrama de navegabilidad del usuario.....68

ÍNDICE DE TABLAS

Tabla 1: Tabla de de servicios (Endpoints) API.php 25

Tabla 2: Catálogo de requisitos 37

Tabla 3: Caso de uso de gestión de identificación 42

Tabla 4: Caso de uso de gestión de deportes 42

Tabla 5: Caso de uso de gestión de pistas 43

Tabla 6: Caso de uso de gestión de usuarios 43

Tabla 7: Caso de uso de gestión de reservas 44

Tabla 8: Caso de uso para crear un nuevo deporte 45

Tabla 9: Caso de uso para ver un deporte 46

Tabla 10: Caso de uso para modificar un deporte 46

Tabla 11: Caso de uso para borrar un deporte 47

Tabla 12: Caso de uso para crear una nueva pista 48

Tabla 13: Caso de uso para ver una pista 49

Tabla 14: Caso de uso para borrar una pista 49

Tabla 15: Caso de uso para ver los usuarios 51

Tabla 16: Caso de uso para borrar un usuario 51

Tabla 17: Caso de uso para crear un nuevo usuario 52

Tabla 18: Caso de uso para modificar los datos de usuario 52

Tabla 19: Caso de uso para crear una nueva reserva 53

Tabla 20: Caso de uso para ver una reserva 53

Tabla 21: Caso de uso para cancelar una reserva 53

1. Introducción

1.1 Resumen extendido

Como hemos comentado previamente, el proyecto que se presenta se trata de realizar el análisis, diseño e implementación de una aplicación web mediante la cual podamos, por una parte, gestionar las reservas del Servicio de Deportes del Campus de la Universidad Francisco de Vitoria y, por otro, darle al usuario la posibilidad de consultar la disponibilidad de las pistas que en ese campus se encuentran y su reserva.

El objetivo del proyecto consiste en desarrollar un sistema de información que cubra con unas necesidades y que consta de los siguientes componentes principales:

- Base de datos
- Módulo de administración que ofrecerá soporte a la BBDD
- Módulo de usuario

Éstos, además, deben cumplir con los siguientes objetivos específicos:

- Proporcionar y definir un Sistema de Información que ayude a conseguir los fines de la organización mediante la definición de un marco estratégico para el desarrollo del mismo.
- Dotar al Servicio de Deportes de la Universidad Francisco de Vitoria de un software que se adapte, y a la vez aproveche, a las nuevas tecnologías permitiéndonos disponer de la información en cualquier lugar y momento y posibilitando las modificaciones correspondientes.
- Facilitar la gestión tanto a administradores como a usuarios mediante paneles de control adaptados a cada uno de ellos.

- Mejora del diseño de la actual página del Servicio adaptándolo a cualquier dispositivo que permita visualizarla.

2. Descripción del problema

2.1. Análisis de situación actual

En este apartado se analizarán dos aspectos fundamentalmente, por un lado el estado actual de la página del Servicio de Deportes de la Universidad Francisco de Vitoria; por otro, se analizarán desarrollos ya existentes de gestión de pistas deportivas, sobre todo para entender cuál es el paradigma actual y funcionalidades tipo que deberían contemplar.

En relación al Proyecto que se explica en esta memoria, se ha de decir que la página actual no dispone de una gestión de reservas en línea desde la que los usuarios pudieran realizarlas y gestionarlas por medio de un panel de control como el que se propone en el mismo. Como al fin y al cabo la página web que se propone tiene capacidad para integrar la antigua página e incluir en ésta todos lo que rodea a la gestión de reservas haremos un análisis de toda la página para ver las mejoras de la misma.

Por otro lado, en cuanto a la valoración que se ha hecho de otros servicios ya existentes, en casi todos ellos nos encontramos con similares funcionalidades (gestión del perfil personal, reserva de pistas, gestión de las reservas). También hemos encontrado alguno que va más allá e incluye funcionalidades como una red social propia, para proponer partidos entre sí, este es el caso concreto de la escuela de pádel y tenis El Estudiante, que facilita este servicio por el cual se fomenta el uso de las instalaciones y al mismo tiempo facilita a aquellos usuarios que no consiguen reunir a 4 personas para jugar un partido de pádel, puedan hacerlo. En nuestro caso optaremos por desarrollar los módulos más comunes.

También es importante destacar que en casi todos los desarrollos existentes ya consultados la experiencia de usuario en entorno móvil es bastante pobre, la navegación

no está del todo adaptada a Smartphone y eso dificulta el uso de las aplicaciones, sobre todo teniendo en cuenta que actualmente más de la mitad de las consultas de este tipo las hacemos desde el móvil, es por ello, que en nuestro desarrollo priorizaremos esa experiencia de usuario en Smartphone, tratando de conseguir un diseño que se adapte también a pc y tablets, pero siempre con la base móvil.

2.2. Objetivos del PFG

El objetivo del proyecto consiste en desarrollar un sistema de información que cubra con unas necesidades y que consta de los siguientes componentes principales:

- Base de datos
- Módulo de administración que ofrecerá soporte a la BBDD
- Módulo de usuario

Éstos, además, deben cumplir con los siguientes objetivos específicos:

- Proporcionar y definir un Sistema de Información que ayude a conseguir los fines de la organización mediante la definición de un marco estratégico para el desarrollo del mismo.
- Dotar al Servicio de Deportes de la Universidad Francisco de Vitoria de un software que se adapte, y a la vez aproveche, a las nuevas tecnologías permitiéndonos disponer de la información en cualquier lugar y momento y posibilitando las modificaciones correspondientes.
- Facilitar la gestión tanto a administradores como a usuarios mediante paneles de control adaptados a cada uno de ellos.
- Mejora del diseño de la actual página del Servicio adaptándolo a cualquier dispositivo que permita visualizarla.

2.3 Metodología

Para llevar a cabo el proyecto, ha sido necesaria la realización de una serie de pasos o etapas las cuales se pueden ver a continuación:

- Análisis de la situación actual del Servicio de Deportes y, en especial, de la gestión de reservas de éste. Al no existir una gestión de reservas online como tal, nos ha facilitado las funciones de creación de requisitos adecuándolos a las necesidades del Servicio.

- Una vez encontrado dicho problema, hemos visitado páginas web con objetivos similares para buscar la mejor solución.

- Se realiza la Planificación General del Proyecto y se decide emplear una metodología ágil para el desarrollo del proyecto, en concreto Scrum [1]. En Scrum un proyecto se ejecuta en bloques temporales cortos y fijos (iteraciones de un mes natural y hasta de dos semanas). Cada iteración tiene que proporcionar un resultado completo, un incremento de producto que sea potencialmente entregable, de manera que cuando el cliente (Product Owner) lo solicite sólo sea necesario un esfuerzo mínimo para que el producto esté disponible para ser utilizado.

- Se realiza la primera de las fases objeto de estudio en el proyecto. Planificación del Sistema de Información, en dicha fase se aprueba que una vez realizada y aprobada ésta no se volverá a dicha etapa por lo que tendremos que definir todo de manera correcta.

- Se analizará el Análisis de Sistema de Información, Diseño del mismo, así como la Construcción y Aceptación. Tras estas etapas, recogeremos la documentación generada.

- Se prepararán y revisarán los documentos y entregables del proyecto con el fin de corregir los distintos errores para la aceptación del proyecto.

- Se comenzará con la parte técnica como tal, comenzando por el diseño de la Base de Datos y, posteriormente, con el desarrollo de la aplicación o sitio web.

2.4. Planificación del proyecto

Al elegir una planificación del proyecto basada en metodología ágil Scrum, una seguimos una estrategia de desarrollo incremental. En lugar de la planificación y ejecución completa del producto, acometiendo una tras otra fase en un ciclo secuencial o de cascada; trabajamos en lograr iteraciones o sprints.

Estas iteraciones se caracterizan en que para poder completar el máximo de requisitos en la iteración, se debe minimizar el número de objetivos/requisitos en que el equipo trabaja simultáneamente, completando primero los que den más valor al cliente. En nuestro caso se ha optado por mostrar una versión inicial de la aplicación con acceso al módulo usuario antes que al módulo administrador, ya que la finalidad primaria de nuestro cliente es obtener una aplicación útil para el usuario.

Esta forma de trabajar, que se ve facilitada por la propia estructura de la lista de tareas de la iteración, permite tener más capacidad de reacción frente a cambios o situaciones inesperadas.

Por tanto, en nuestro caso además de definir un plan de proyecto con etapas genéricas y que veremos más adelante, se han priorizado las tareas dentro de ellas, generando así una serie de sprints a acometer. Estos serían los principales sprints o iteraciones del proyecto:

- Arquitectura de base de datos
- Definición de endpoints de la apirest
- Definición de la estructura de la aplicación
- Configuración de entorno de desarrollo
- Automatización de tareas
- Configuración de deploys

- Creación de api rest y sus endpoints
- Creación del proyecto base
- Implementación de módulos:
  - Registrar usuario
  - Login usuario
  - Persistencia de datos del login
  - Cerrar sesión
  - Tipos de deportes
  - Pistas por deporte
  - Calendario y gestión de fechas
  - Reserva de pistas
  - Anulación de pistas
  - Ficha de usuario
  - Listado de reservas
  - Módulo administrador:
    - Crear usuarios administrador
    - Anular cualquier reserva
    - Listar y borrar usuarios
    - Editar información de deportes y pistas
    - Borrado de deportes y pistas
- Estilos css al esqueleto html
- Testing
- Memoria

Para la planificación del proyecto se ha definido un calendario de trabajo en conjunción con el tutor del proyecto, para poder avanzar en cuanto a resultados obtenidos, desviaciones de trabajo o mejoras indicadas. En este calendario se han cerrado una serie de reuniones ajustadas en el tiempo para comprobar los avances tanto en la parte técnica como en la parte de definición del trabajo realizado que queda plasmado en esta memoria.

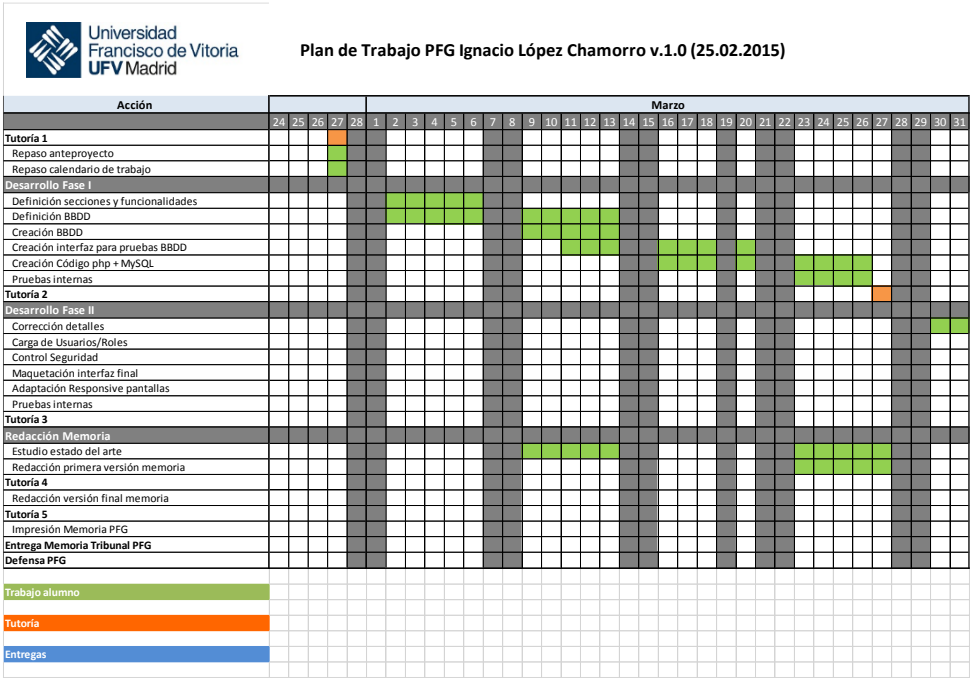

Figura 1: Plan de trabajo Marzo

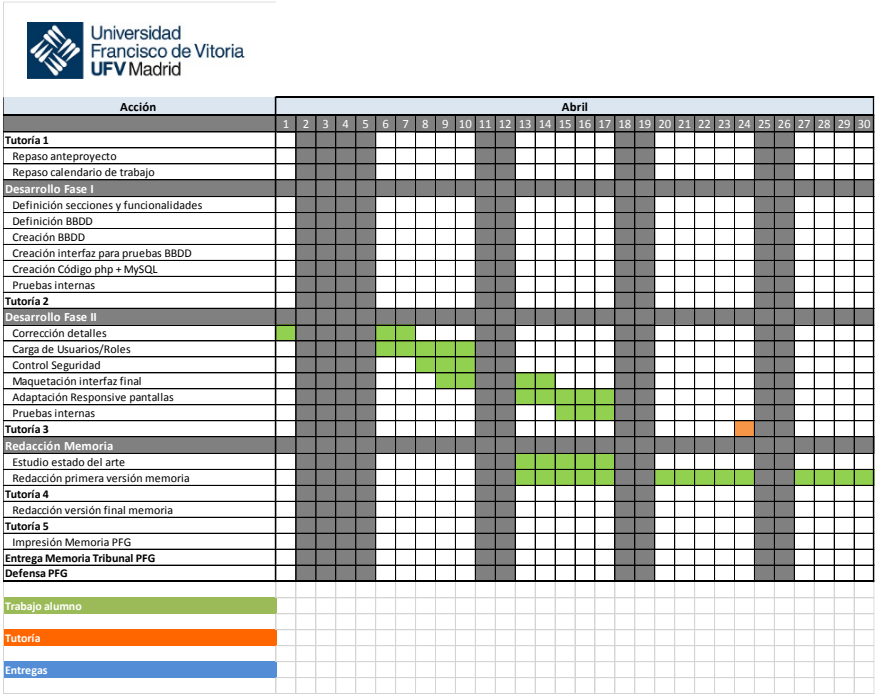

Figura 2: Plan de trabajo Abril

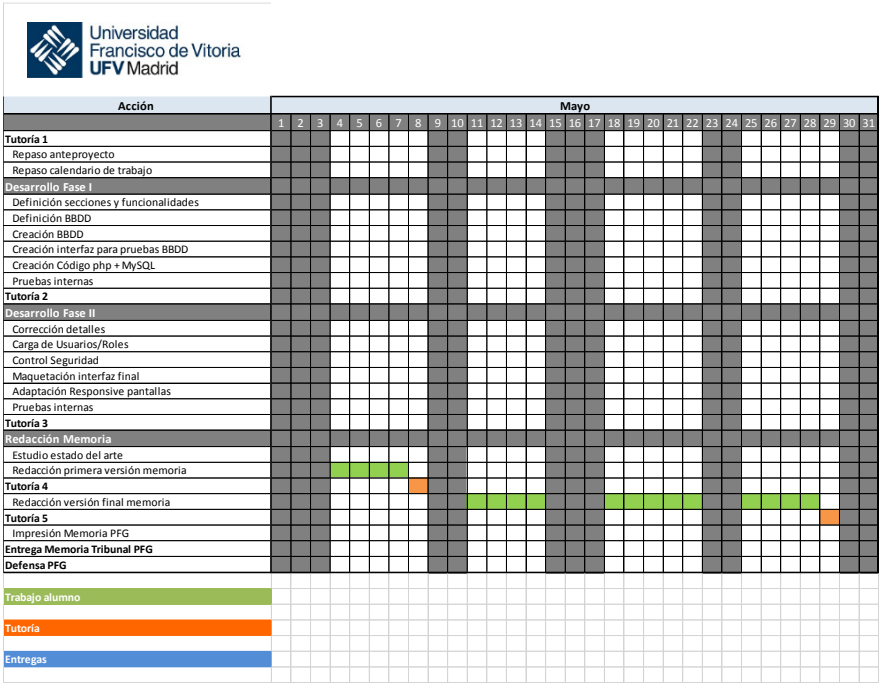

Figura 3: Plan de trabajo Mayo

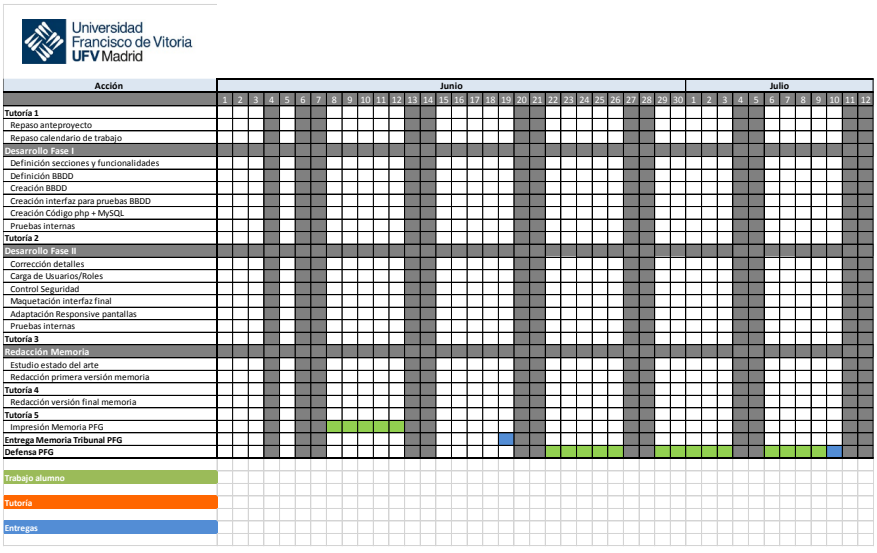

Figura 4: Plan de trabajo Junio-Julio

2.5. Restricciones

Las restricciones con las que nos encontramos en cualquier proyecto a priori podrían ser muchas, pero en esta ocasión en la que el propio cliente somos nosotros mismos, dichas restricciones quedan minimizadas en dos. En primer lugar, nos encontramos con una restricción en cuanto a desarrollos tecnológicos ya existentes, teniendo que adecuar el nuestro a ese paradigma ya existente. Con esto nos referimos a restricciones en cuanto a diseños y adaptación de las pantallas al "estilo" corporativo de la UFV.

En segundo, lugar, podríamos destacar las restricciones derivadas de la integración del servicio de reservas de pistas dentro del ecosistema web de la propia UFV, nos referimos en concreto a la conexión la base de datos de la UFV. En este caso se ha optado por no hacer consulta contra dicha base de datos ya que no tenemos acceso a la misma, pero de cara a una integración final sería importante valorarlo.

3. Detalle del proyecto:

3.1. Identificación de requisitos

Para no repetir funciones que son idénticas en distintos roles, primero mostraremos los distintos procesos según el rol [2] y, finalmente, analizaremos las necesidades de cada uno de ellos. Posteriormente se explicarán en detalle todos los servicios desarrollados y sus funcionalidades.

En el siguiente diagrama podemos ver los procesos permitidos cuando el usuario activo tiene rol de administrador:

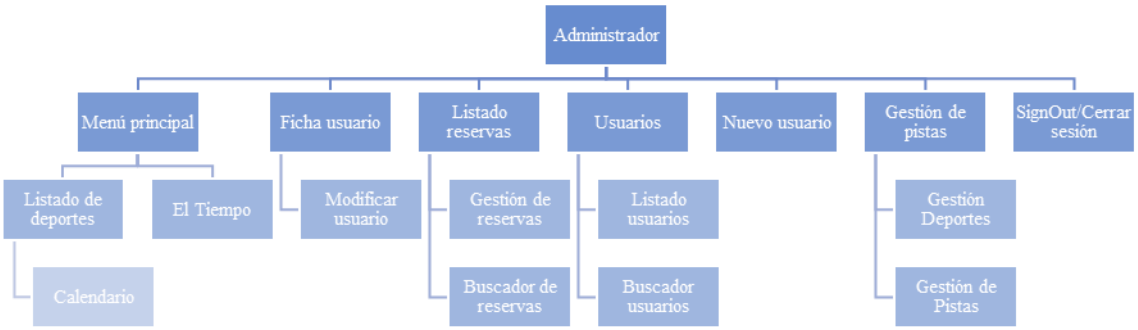

Diagrama 1: Procesos competencia de los administradores

Y, por último, en el siguiente diagrama podemos ver los procesos permitidos cuando se trata de un usuario que se registra en la página:

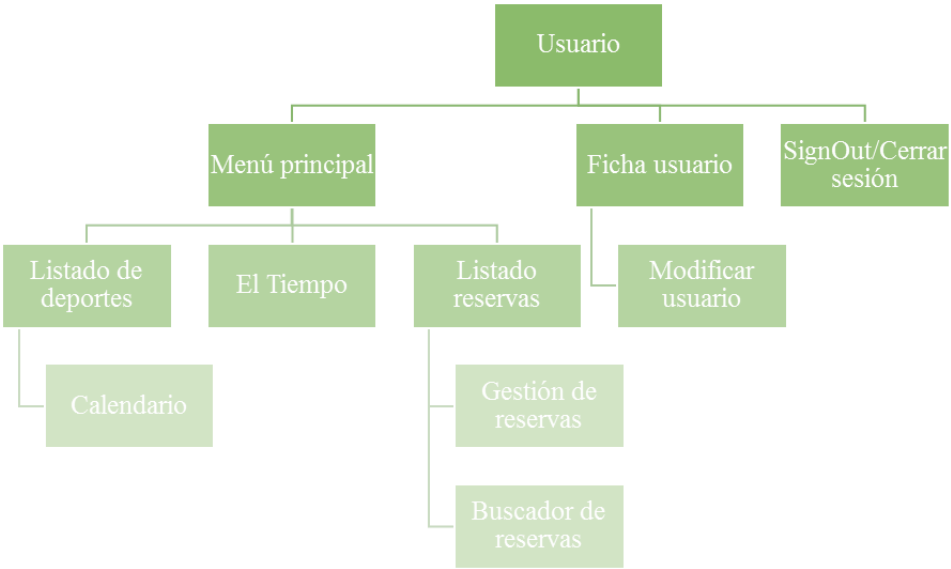

Diagrama 2: Procesos competencia de los usuarios

A continuación se analizarán las necesidades de cada una de las tareas que componen los modelos de procesos anteriores:

• Nuevo deporte: únicamente los administradores tendrán la posibilidad de gestionar los deportes que los usuarios del Servicio de Deportes podrán practicar en el campus.

• Ver deportes: los administradores tendrán la posibilidad de ver los deportes almacenados en la base de datos. Se realiza una consulta a la base de datos que envía la información de cada uno de los deportes.

• Modificar deporte: los administradores podrán modificar los deportes ya creados a partir de un formulario donde le aparecerán los datos que en ese momento están almacenados y podrán dejarlos o no igual.

• Borrar deporte: los administradores serán los únicos que podrán borrar alguno de los deportes ya creados.

• Nueva pista: tras la creación de un deporte por parte del administrador, es éste el que tendrá la posibilidad de crear pistas del mismo.

• Ver pistas: al igual que en el punto anterior, es el administrador el que tendrá los permisos para ver el listado con todas las pistas creadas.

• Borrar pistas: serán los administradores los encargados de borrar las pistas cuando lo crean conveniente.

• Nueva reserva: en este caso, podrán realizar una nueva reserva tanto el administrador, como el propio usuario. Al mostrar la disponibilidad de la pista, se aparecerán las horas disponibles, no obstante para evitar conflictos de una reserva simultánea, siempre que se permite la reserva se realizará una consulta a la base de datos con la el horario y deporte deseado por el usuario y según la disposición podrá o no podrá realizarse dicha reserva. En caso afirmativo insertaremos la información en la base de datos.

• Ver reservas: al igual que en el anterior, administrador y usuario podrán ver las reservas. Una única diferencia: el administrador podrán ver absolutamente todas las reservas y los usuarios, lógicamente, sólo podrán ver las reservas que hayan realizado.

• Cancelar reserva: podrán cancelar reservas el administrador y usuario. El administrador podrá cancelar cualquier reserva, mientras que el usuario sólo tendrá acceso a sus reservas.

• Ver usuario: el administrador podrá ver la información de uno o de todos los usuarios que estén registrados en nuestra base de datos. Deberemos comprobar la no duplicidad de esta información para así mantener la consistencia de la misma.

• Borrar usuario: los administradores podrán eliminar usuarios de la base de datos.

• Modificar datos: se trata de una acción que podrá realizar el usuario y el administrador. Le aparecerá la información que existe en el momento de la modificación en la base de datos y simplemente tendrá que sustituirla por la nueva y finalmente se insertará en la misma.

• Validación y verificación: Con el objetivo de no insertar información repetida o datos incorrectos (estructura, forma, etc.) antes de insertar los valores en la base de datos haremos ciertas comprobaciones de la información que estamos mandando.

• Transacción: se realizará una inserción, modificación o borrado en la base de datos tras pasar el anterior punto de validación y verificación.

Análisis de las Necesidades de Información

En primer lugar, vamos a ver el análisis de las necesidades de información por parte del sistema:

• Validación / Verificación: El sistema validará los datos de uno o de varios empleados antes de realizar una o varias transacciones en la base de datos. Una vez validados estos y realizadas la o las transacciones se verificará que los datos de la base de datos son los deseados.

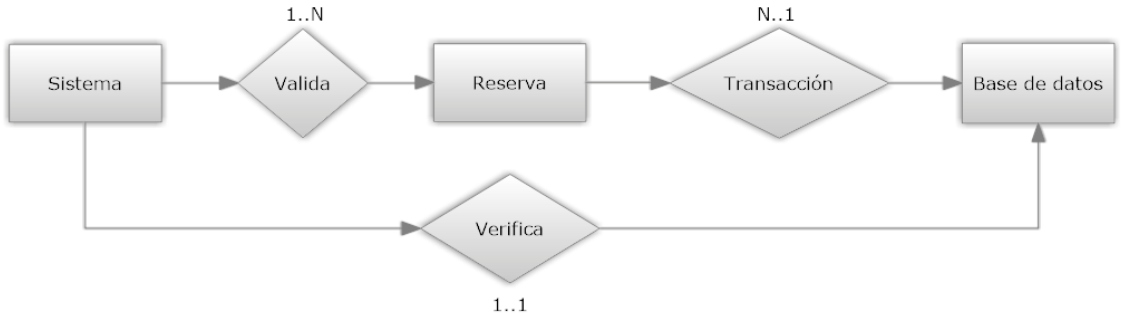

Diagrama 3: E-R de la validación o verificación de datos

• Transacción de la base de datos: El sistema realizará una transacción en la única base de datos que tenemos.

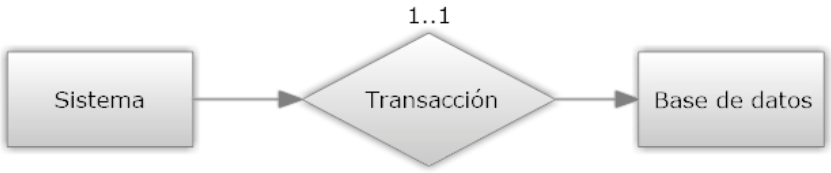

Diagrama 4: E-R de transacción en la base de datos

A continuación, vamos a ver el análisis de las necesidades de información del administrador:

• Ver usuario: el administrador buscará en la base de datos uno o varios usuarios los cuales serán mostrados en un listado.

Diagrama 5: E-R para ver usuarios

• Nuevo deporte: el administrador creará uno o varios deportes que serán insertados en nuestra base de datos.

Diagrama 6: E-R de creación de un deporte

• Borrar deporte: el administrador consulta con la base de datos que le muestra los deportes que haya y del cual seleccionará uno que será borrado de la propia base de datos.

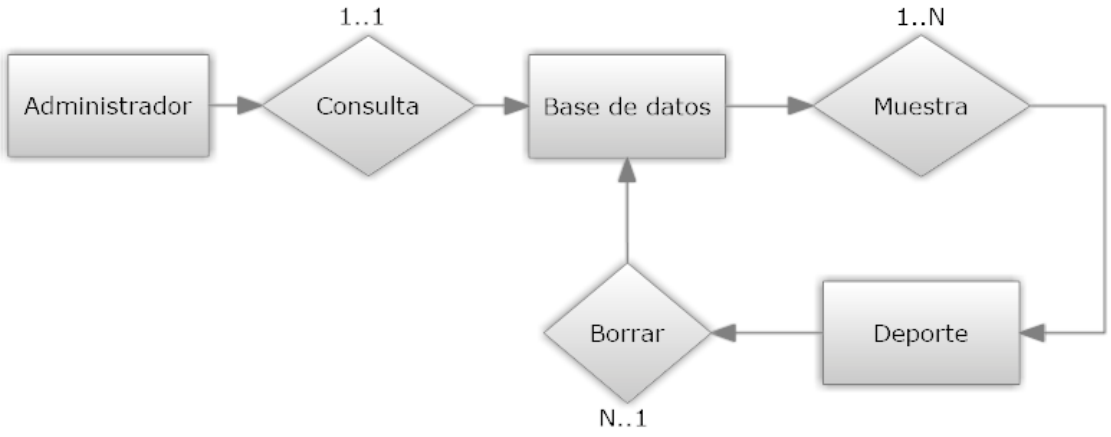

Diagrama 7: E-R para borrar un deporte

• Nueva pista: el administrador creará una o varias pistas que serán insertadas en nuestra base de datos.

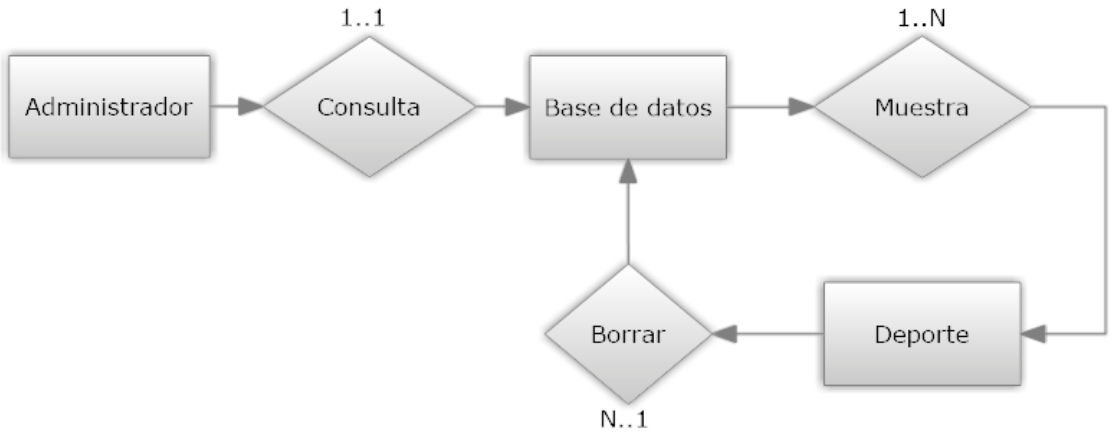

Diagrama 8: E-R de creación de una nueva pista

• Ver pistas: el administrador buscará en la base de datos una o varias pistas las cuales mostradas en un listado.

Diagrama 9: E-R para ver las pistas

• Nueva reserva: el administrador creará una o varias reservas que serán insertadas en la base de datos.

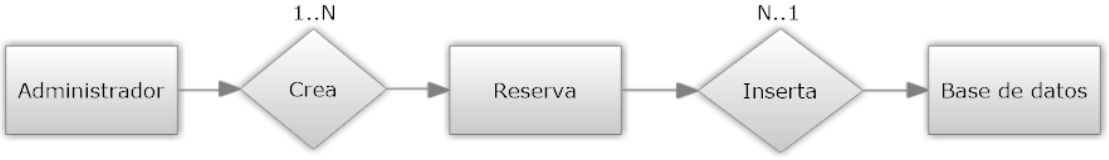

Diagrama 10: E-R de creación de reserva

• Ver reservas: el administrador buscará en la base de datos una reserva la cual será mostrada.

Diagrama 11: E-R para ver reservas

• Cancelar reserva: el administrador buscará en la base de datos una o varias reservas de las cuales podrá borrar las que él desee.

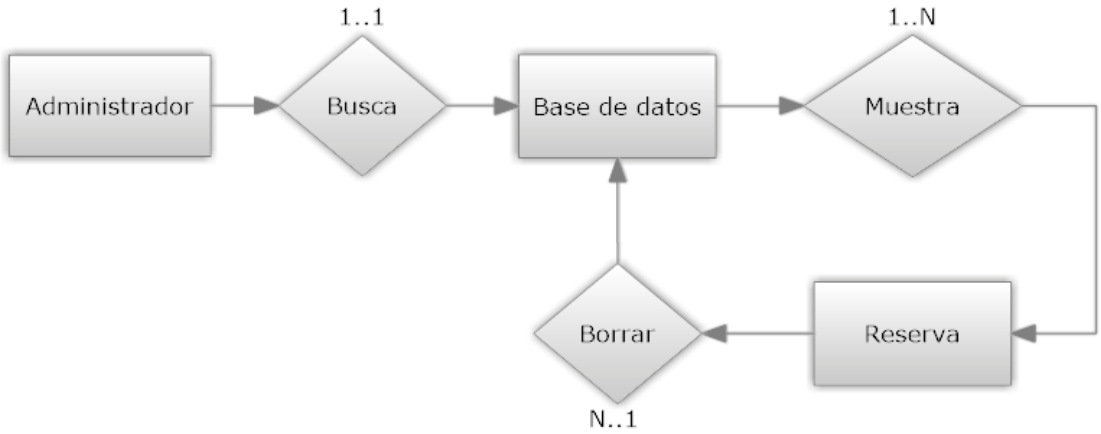

Diagrama 12: E-R de cancelación de una reserva

Seguidamente, vamos a ver el análisis de las necesidades de información del usuario final del Servicio de Deportes:

• Nueva reserva: el usuario creará una o varias reservas que serán insertadas en la base de datos.

Diagrama 13: E-R de creación de una nueva reserva

• Ver reserva: el usuario buscará en la base de datos una reserva la cual será mostrada.

Diagrama 14: E-R para ver una reserva

• Modificar datos: el usuario buscará sus datos en la base de datos que los mostrará y que tras modificarlos se guardarán en la propia base de datos.

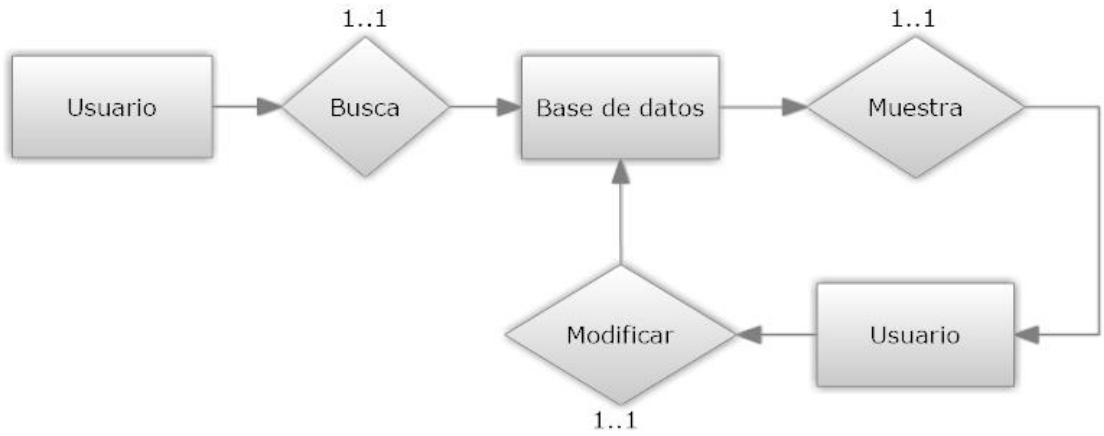

Diagrama 15: E-R de modificación de datos personales

3.2. Definición de la arquitectura tecnológica

Se ha optado por una arquitectura separada principalmente en tres capas lógicas:

- Modelo de Datos
- API-RESTful
- Front-end

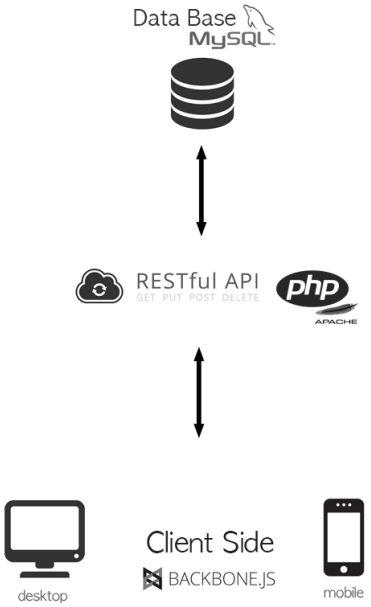

Figura 5: Definición arquitectura tecnológica

El modelo será una base de datos relacional MySQL [3], [4] que guardará todos los datos necesarios para el funcionamiento de la aplicación. Se creará una capa intermedia a modo broker cuyo único fin es comunicar el front-end con el modelo de datos que será la API RESTful [5], [6], [7]. Y una tercera capa que será el front-end donde estará el peso de la lógica de negocio y las vistas para que los usuarios y administradores puedan interaccionar con el modelo.

Se ha optado por esta arquitectura para que los costes de servidores sean los mínimos posibles llevando la lógica de negocio al front-end gracias a la potencia que nos brinda el framework Backbone [8] desarrollado en JavaScript que pasará a ser ejecutado en el cliente. Actualmente esto es viable gracias al aumento de la potencia media de todos los dispositivos con los que cuentan los usuarios, de manera que conseguimos rebajar el número de procesos en el servidor prácticamente al mínimo reduciendo el coste notablemente. Se entiende que ante picos de uso y la necesidad de poder dar servicio a millones de personas de forma concurrente esta misma arquitectura estará preparada para ser desplegada en un entorno de alta disponibilidad con escalado automático sin tener que realizar ninguna refactorización de código.

Para el desarrollo y en concordancia con la reducción de costes se ha optado por utilizar solo software libre exento del pago de licencias.

Seguidamente se van a definir las tecnologías de cada capa.

MODELO DE DATOS:

Será un modelo de datos relacional en MySQL, MySQL es la base de datos de código abierto más popular del mundo. Trabaja mediante un sistema de gestión de bases de datos relacional, con multitud de hilos y disponible para varios usuarios simultáneamente multihilo MySQL se ofrece bajo la licencia GNU GPL para cualquier uso compatible con esta licencia, o para las compañías que quieran trabajar con ello, la posibilidad de licencias privativas.

Actualmente gran parte de grandes proyectos web están desarrollados con una base de datos MySQL, tal es el cas de Facebook, Twitter, Wikipedia o YouTube, incluso Google la usa en determinados desarrollos.

Se ha instaurado la sentencia DELETE ON CASCADE con los deportes, usuarios y pistas. Al eliminar un usuario se eliminan sus reservas. Si se elimina un deporte se eliminan sus pistas y sus alquileres asociados. Esto es para evitar dejar entidades colgadas

que se puedan mostrar, por tanto, si a nivel de negocio se desean eliminar determinados datos pero se quiere guardar esa información, es importante almacenarla previamente.

Se realiza un back-up diario de la base de datos de forma automática.

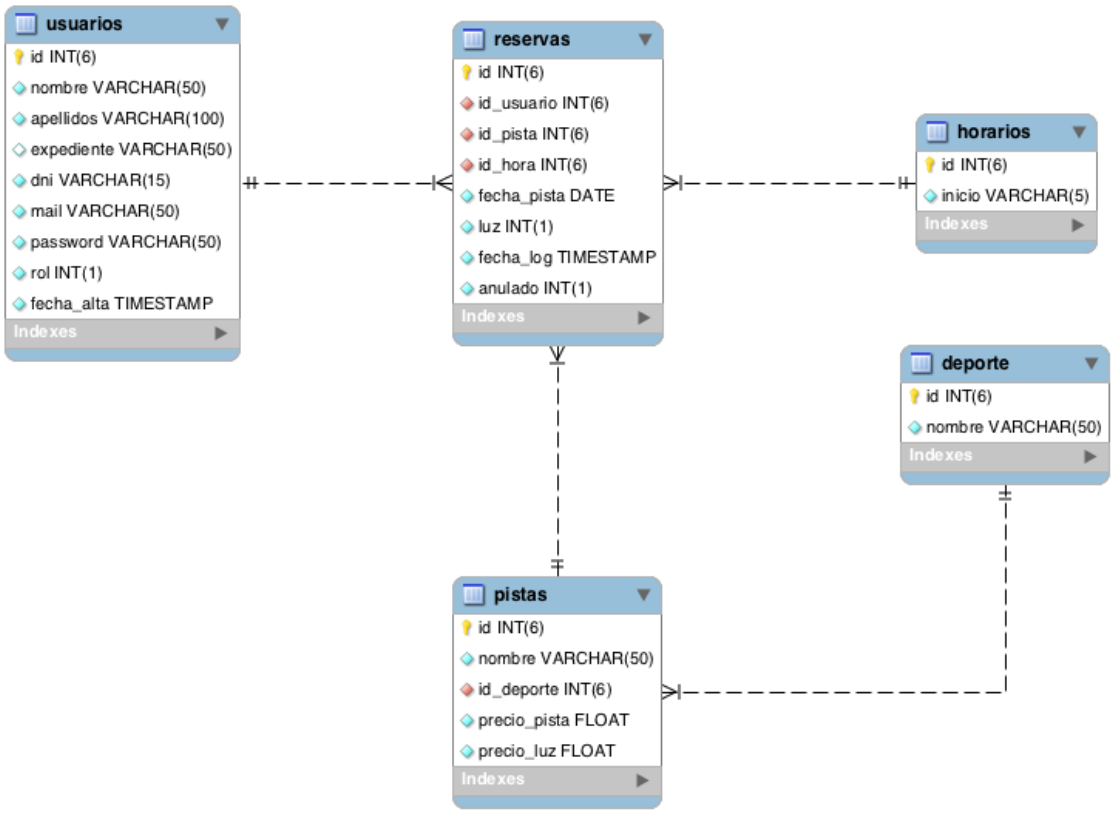

Figura 6: Modelo Entidad-Relación BBDD

API-RESTFUL:

Actuará de broker ejerciendo la comunicación entre el frontend y el modelo de datos. Esta API no guardará datos en memoria por lo que no se encargará de la persistencia siendo su único fin la gestión de recuperación de datos y empaquetarlos como JSON para devolverlos al frontend. Se crea esta capa intermedia para garantizar la seguridad de los datos, de forma que el frontend nunca tendrá acceso directo a la base de datos siendo trabajo de este api el lanzamiento de las consultas en un entorno controlado.

Las variables que proceden del frontend serán parametrizadas siempre evitando así cualquier ataque tipo SQL Injection.

Estará desarrollada en PHP orientado a objetos y abstraído en dos clases, Rest.php y API.php que extiende de Rest. Para generar las consultas se utilizará la librería de php PDO. Estará preparada para trabajar las consultas con urls amigables.

Su lógica implica que por cada petición vamos a recibir un objeto JSON ya sea dando la información solicitada o bien informando del error que ocurra por ejecución, petición no aceptada, datos incorrectos o falta de privilegios para dicha consulta.

Clase Rest:

Esta clase se ocupa de dos tareas principalmente:

- Devolver las cabeceras con el código de estado y el resultado de la petición al cliente.
- Filtrar los datos enviados en la petición.

A) El método mostrarRespuesta recibe los parámetros $data, que contiene la respuesta JSON a enviar al cliente, y $estado que especifica el código de estado HTTP que acompañará a la respuesta. Este método se encarga de asignar el código de estado con el que se configurarán las cabeceras. Configurará las cabeceras que se van a enviar junto con la respuesta, mediante la llamada al método setCabecera. Y mostrará dicha respuesta.

B) El método setCabecera crea dos cabeceras que acompañarán a la respuesta de la petición. Para ello utilizará el código de estado asignado en el método mostrarRespuesta y la descripción del código obtenida mediante el método getCodEstado. Estas cabeceras no serán enviadas hasta que no se envíe la respuesta en mostrarRespuesta con la instrucción echo $data.

C) El método getCodEstado contiene un array asociativo donde las claves son los posibles códigos de estado y los valores son las descripciones asociadas a esos códigos. Por lo tanto, a partir del código de estado que se enviará junto a las cabeceras y la respuesta, devolverá su descripción.

Los métodos encargados de limpiar los datos se encargan de sanear los datos que acompañarán a las peticiones GET, POST, PUT y DELETE, y son los siguientes:

A) El método tratarEntrada se encarga de sanear el array datos de entrada llamando al método limpiarEntrada y asigna dicho array de datos al atributo $datosPetición. Para ello primero comprueba cual es el método de petición ($_SERVER['REQUEST_METHOD']) y pasa los datos del array superglobal de la petición a limpiarEntrada. Esto quiere decir que si el método de entrada es GET se tratarán los datos del array $_GET. Y si es POST se tratarán los datos que puedan estar contenidos en $_POST.

B) El método limpiarEntrada se encarga de sanear los datos que se le pasen como parámetro. Es un método recursivo para tratar cada uno de los valores de un array.

Htaccess:

Como comentamos anteriormente, queremos que el servicio pueda ser utilizado mediante URL's amigables. Para ello deberemos de especificar una regla de reescritura de URL's en un fichero .htaccess. Que lo definiremos en la raíz del proyecto.

Con esta regla de reescritura estamos especificando:

- El directorio base es /login_restful/
- Se han añadido tres condiciones para restringir la reescritura sólo a rutas que no existan previamente. Es decir, que no valdría realizar reescritura, por ejemplo, para www.dominio.com/img/img.png (suponiendo que esta ruta y recurso existe).

- La primera condición previene los directorios que ya existan con la bandera !-d.
- La segunda condición hace que se ignoren ficheros que ya existan con la bandera !-f.
- Y la tercera condición hace que se ignoren los enlaces simbólicos que ya existan con !-l.

- Luego con la regla de reescritura transformaremos una URL amigable a una URL con la que el servidor pueda trabajar. Por lo tanto http://pfc.ilopezchamorro.com/api/borrarUsuario/1 será transformado a http://pfc.ilopezchamorro.com/api/Api.php?url=borrarUsuario/1. Ya que /borrarUsuario/1 será tomado como referencia $1.

Clase API:

Anteriormente definimos la clase Rest y vamos a utilizarla como clase base a heredar. De esta forma tendremos disponibles tanto sus métodos como atributos. La clase que vamos a definir a continuación (Api) va a ser la encargada de:

- Conectar con la base de datos
- Determinar que método debe de utilizar para resolver la petición, a partir de la url de dicha petición.
- Utilizar los métodos heredados de la clase padre para sanear los datos de entrada de la petición y devolver el resultado al cliente.

A) El método conectarDB como su propio indica, se encarga de conectar con la base de datos que contendrán los recursos del servicio. Para ello utilizaremos PDO como librería de abstracción para tratar con la base de datos MySQL.

B) El método devolverError utiliza un identificador recibido como parámetro para devolver un array asociativo con dos elementos: elemento 'estado' con valor error y elementos 'msg' con la descripción de error.

C) El método procesarLLamada es uno de los métodos más importantes ya que es el encargado de procesar la URL de la petición y a partir de ella se llamará al método que resolverá la citada petición. Como hemos comentado en el apartado anterior, si se hace una petición a la URL http://pfc.ilopezchamorro.com/api/borrarUsuario/1, esta URL será transformada internamente a http://pfc.ilopezchamorro.com/api/Api.php?url=borrarUsuario/1. Por lo tanto, si obtenemos el valor de la variable global 'url' ($_REQUEST['url']) obtendremos borrarUsuario/1. Y esto nos puede sugerir que necesitamos invocar al método borrarUsuario con el 1 como argumento para resolver la petición. Evidentemente no todas las URL estarán asociadas a llamadas a métodos con argumentos. Por ejemplo: http://pfc.ilopezchamorro.com/api/usuarios.

Los siguientes métodos de la clase son los encargados de realizar cada una de las operaciones asociadas con las peticiones.

A) El método usuarios comprueba si se ha accedido a él mediante una petición GET y devuelve un objeto JSON con los datos de los usuarios del servicio.

B) El método login comprueba si se ha accedido a él mediante una petición POST, comprueba si lo datos que acompañan a dicha petición ($datosPeticion) son adecuados y devuelve un objeto JSON indicando si el usuario existe.

C) El método actualizarNombre($id) tras comprobar que se ha accedido a él con una petición PUT, utiliza los datos pasados junto a dicha petición y el parámetro $id, extraído de la URL en procesarLLamada, para actualizar el nombre de un usuario existente (con dicho identificador). Posteriormente devuelve un objeto JSON confirmando que se ha actualizado los datos de un usuario.

D) El método borrarUsuario($id) tras comprobar que se ha accedido a él con una petición DELETE, utiliza el parámetro $id, extraído de la URL en procesarLLamada, para borrar el usuario con dicho identificador. Posteriormente devuelve un objeto JSON confirmando que se ha borrado el usuario.

E) Finalmente el método crearUsuario tras comprobar que se ha accedido a él mediante una petición POST, utiliza los datos pasados junto a dicha petición para crear a un nuevo usuario. Posteriormente se devuelve un objeto confirmando la creación del usuario.

Finalmente se creará una instancia de la clase Api y se llamará al método procesarLLamada. Esto es necesario ya que cada vez que se realice una petición a una URL del servicio, gracias a la regla de reescritura se llama al script Api.php. Y de esta manera se creará el objeto Api y se invocará al método que procesa la URL y llama al encargado de resolver la petición.

Hasta ahora se ha documentado cómo funciona la api, a continuación se documentará cada endpoint del servicio

Método	URL	Descripción	Parámetros
POST	nuevoUsuario/	Crea un nuevo Usuario	nombre, apellidos, expediente, dni, password, mail
POST	nuevoAdmin/	Crea un nuevo Adminsitrador	nombre, apellidos, expediente, dni, password, mail, rol
POST	actualizarUsuario/	Actualiza los datos de un user	nombre, apellidos, expediente, dni, password, mail
POST	actualizarAdmin/	Actualiza los datos de un admin	nombre, apellidos, expediente, dni, password, mail
POST	eliminarUsuario/	Elimina un usuario	id_usuario
POST	nuevaHora/	Crea una nueva hora a todas las pistas	inicio
POST	nuevoDeporte/	Crea un nuevo Deporte	nombre
POST	EliminarDeporte/	Elimina un deporte	id_deporte
POST	ModificarDeporte/	Modifica un Deporte	id, new_name_deporte
GET	deportes/	Lista los deportes con Pistas asociadas	null
GET	deportesAdmin/	Lista todos los deportes	null
POST	pistas/	Lista las pistas de un deporte de un día determinado	id (pista), fecha_pista

POST	modificarPistas/	Modifica una pista	id (pista), nombre, precio_luz, precio_pista
POST	eliminarPista/	Elimina una pista	id (pista)
POST	nuevaPista/	Crea una pista asociada a un deporte	nombre, id_deporte
POST	pistasAdmin/	Lista todas las pistas	id (pista)
POST	reserva/	Crea un nueva Reserva	id_usuario, id_pista, id_hora, fecha_pista, luz, anulado
POST	reservasUsuario/	Lista todas las reservas de un usuario	id_usuario
GET	reservasAdmin/	Lista todas las reservas de todos los usuario	null
PUT	anularReserva/	Anula una reserva	id_reserva
GET	usuarios/	Lista los usuarios	null
POST	login/	Confirma credenciales del usuario	mail, password

Tabla 1: Tabla de de servicios (Endpoints) API.php

FRONT-END:

El front-end está compuesto por un conjunto de tecnologías web de última generación.

- HTML5
- CSS3
- JavaScript (lógica de negocio)

Aunque estas tecnologías existen casi desde el comienzo de la web sufren continuas actualizaciones y cambios sobre sus estándares mejorando su rendimiento y ampliando sus capacidades.

Conviene destacar un poco la historia de JavaScript que nació para dar interactividad a las webs estáticas. Fue Microsoft con Internet Explorer 5.5 quien sin saberlo dotó de vida a JavaScript creando los objetos XHR (comunicación Ajax) aunque la comunidad no supo aprovechar la potencia de este objeto hasta varios años después.

Este estándar nos permite hoy en día comunicar JavaScript directamente con el servidor a través de llamadas Ajax de manera asíncrona (sin tener que recargar la página). Gracias a esto se han desarrollado frameworks como Backbone que ha supuesto una revolución en el desarrollo web, siempre se ha tenido JavaScript como un lenguaje menor por que no se ha sabido aprovechar su naturaleza funcional; gracias a que son objetos de primera clase por que se pueden pasar funciones anónimas como parámetro o devolver funciones entre otras cualidades. Backbone nos ayuda a tener un patrón de diseño MVC en front teniendo la capacidad de desarrollar código bien estructurado y abstraído que nos permite tener un código limpio, escalable y de fácil mantenimiento.

La aplicación será tipo 'ONE-PAGE' con un único archivo .html que será el index y nunca se recargará la página ya que la gestión el peso de la aplicación lo lleva Backbone que será capaz de cambiar dinámicamente las url gracias al push-state de HTML5 que permite añadir parámetros a la url tras el hash ´#´, este lo escondemos para dejar una url más limpia. Véase como ejemplo en los comienzos Twitter que usa esta tecnología sus urls estaban formadas de la siguiente manera: twitter.com/!#/user, actualmente y gracias a HTML5 se puede esconder este valor.

Backbone inyectará partes de HTML dentro del index de manera asíncrona según vayan ocurriendo eventos como pulsar en un botón o bien al cambiar la url sin necesidad de recargar la página.

A continuación pasamos a describir cada punto de manera más extensa.

HTML5

Es la última evolución de la norma que define HTML. En este proyecto al ser tipo ONE-PAGE se encontrará un único archivo .html que será el esqueleto base para generar las vistas de usuario y su gestión se delegará a Backbone.

En este archivo se incluyen las llamadas a los documentos tanto JavaScript como css.

CSS3

CSS u hojas de estilo en cascada (en inglés Cascading Style Sheets) es un lenguaje usado para definir la presentación de un documento estructurado escrito en HTML. El W3C (World Wide Web Consortium) es el encargado de formular la especificación de las hojas de estilo que servirán de estándar para los agentes de usuario o navegadores.

Nuestro CSS será un único archivo minificado y concatenado generado por el precompilador Stylus (se describirá en la parte de desarrollo).

Se ha usado la librería Normalize a modo reset de estilos. El motivo no es otro que cada navegador aplica las reglas por defecto que le parecen más óptimas tanto para HTML5 como para CSS3, esto hace que de partida y sin un reset que iguale los estilos predefinidos de todos los navegadores nos encontramos algunos problemas como que Microsoft aplica bordes a las tablas ya que considera que una tabla debe mostrarse así mientras que Chrome lo muestra sin líneas, salvo que se le indique lo contrario. Esto occurre con otros tantos estilos como márgenes, padings, etc.

Tipografía:

El único recurso externo que se usa es la fuente Roboto para utilizarla en toda la página.

Llamada html:

<link href='http://fonts.googleapis.com/css?family=Roboto:400,300italic,300,100italic,100,400italic,500,900italic,700italic,900,700,500italic' rel='stylesheet' type='text/css'>

JavaScript

Existen librerías que son dependencias para el desarrollo pero que no afectan a la ejecución del proyecto y las explicaremos en un punto posterior sobre el entorno de desarrollo y las automatizaciones de tareas.

A continuación se listarán las librerías de las que el proyecto es dependiente para su ejecución:

- Backbone.js [8]: Core de la aplicación, lleva MVC al frontend
- Underscore.js [9]: Manejo de Objetos con orientación funcional y dependencia de Backbone
- HandleBars.js [10]: Templating que abstrae la lógica de la presentación
- Moment [11]: Parsea, manipula y muestra fechas
- Sha1 [12]: Creación de algoritmo de cifrado Sha1 para la contraseña de los usuarios
- Backbone.localstorage [13]: Librería que facilita el uso de localstorage de HTML5 desde JavaScript
- jQuery [14]: Librería cross-browsing y dependencia de Backbone para peticiones Ajax
- JQuery UI [15]: Capa de Front interactiva usada para el calendario

El resultado final del JavaScript del proyecto será un único JavaScript concatenado, ofuscado y minificado en el archivo ./js/app.min.js con todas las dependencias de producción así como la lógica desarrollada. Gracias a esto conseguimos que toda la lógica que necesita la aplicación esté contenida en una única petición al servidor optimizando así la carga de la aplicación que podría llegar a ser servida de forma distribuida por un CDN tipo Amazon S3 y optimizar aún más el proyecto, aunque no se contempla en esta primera fase armar una arquitectura de alta disponibilidad.

3.3. Entorno de desarrollo

El entorno de desarrollo se ha creado orientado a la automatización de tareas. Esto conlleva una inversión de tiempo inicial muy notable pero nos brinda la seguridad de la eliminación de errores humanos durante los procesos de deploy o desarrollo u inyección de dependencias.

3.3.1. Dependencias de desarrollo

A continuación listamos todo el software y librerías utilizadas para el proceso de desarrollo del proyecto.

- SublimeText: IDE de desarrollo
- Git [16]: Control de Versiones
- PhpMyAdmin: Gestor de Bases de Datos
- Nodejs [17]: Chrome's JavaScript runtime
- Grunt [18]: Ejecutor de Tareas programadas
- grunt-browserify: Inyección de dependencias de Javascript
- Grunt-contrib-clean: Borra directorios
- Grunt-contrib-copy: Copia archivos entre directorios
- Grunt-contrib-cssmin: Minifica los archivos css
- Grunt-contrib-stylus: Precompilados de stylus css
- Grunt-contrib-uglify: Minifica y concatena archivos javascript
- Grunt-open: Abre un directorio en el navegador
- Grunt-ssh: Abre una conexión con el servidor para la subida de archivos por sftp o ssh
- Grunt-replace: Sustituye strings de archivos mediante expresiones regulares
- Grunt-contrib-watch: Observa cambios sobre los archivos para recompilar automáticamente

3.3.2. Control de Versiones

El control de versiones es una herramienta usada para registrar los cambios realizados sobre un archivo o grupo de archivos a lo largo del tiempo, de modo que siempre se puedan recuperar versiones específicas más adelante.

Este sistema te permite volver a a versiones previas de tus archivos, volver a la versión anterior de un proyecto completo, contrastar cambios a lo largo del tiempo, ver el detalle de quién modificó por última vez algo que puede estar ocasionando un problema, qué persona introdujo un error y cuándo, y mucho más. Usar un Sistema de Control de Versiones también significa que si desconfiguras o borras archivos, se pueden recuperar fácilmente. Además, el coste de oportunidad es mínimo.

La característica diferenciadora de Git frente a sus competidores (Subversion y compañía incluidos) es el modo en que Git modela sus datos. A nivel conceptual, casi todos los Sistemas de Control de Versiones almacenan la información como una lista de cambios en los archivos. Estos sistemas (CVS, Subversion, Perforce, Bazaar, etc.), es decir, como un conjunto de archivos y junto con las modificaciones hechas sobre cada uno de ellos a lo largo del tiempo.

En su lugar Git, modela sus datos como un grupo de instantáneas de un mini sistema de archivos. Cada vez que se confirma un cambio, o se almacena el estado de un proyecto, Git toma una instantánea del aspecto de todos los archivos en ese momento, y guarda una referencia a esa foto. Además, para no duplicar información, si hay archivos que no han sido modificados, Git no los almacena de nuevo, sino que crea un puntero que apunta al archivo anterior, como se muestra en la siguiente imagen.

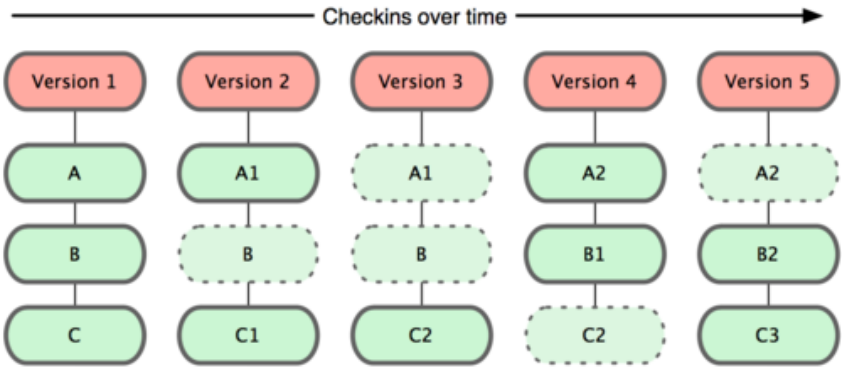

Figura 7: Modelo control de versiones Git[16]

Esta es la principal característica que define la forma de trabajar de Git frente a otros Sistemas de Control de Versiones

Integridad de Git:

Toda la información que Git maneja, es verificada a través de un valor otorgado a la suma de comprobación antes de ser almacenado, a partir de ese momento, dicha suma (checksum) será la identificación de esa información dentro de Git. Esto implica que ante cualquier cambios, se creará una nueva suma y al compararla con la versión actual se detectarán las diferencias. Al integrar este algoritmo en todos sus procesos, Git garantiza que no se pierda información durante la transmisión de la misma y la no corrupción de la información sin ser detectada.

El algoritmo que emplea Git para generar esta suma de comprobación es conocido como la función hash SHA-1. Dicha suma está compuesta por una cadena de caracteres hexadecimales de longitud 40, es decir, 40 caracteres del tipo (0-9 y a-f). Debajo podemos observar un ejemplo de cadenas SHA-1:

24b9da6552252987aa493b52f8696cd6d3b00373

Se pueden observar estos valores hash por todos lados en Git, ya que los usa con mucha frecuencia. De hecho, Git guarda todo no por nombre de archivo, sino en la base de datos de Git por el valor hash de sus contenidos.

Estados de Git:

Git tiene tres estados principales en los que se pueden encontrar tus archivos: confirmado (committed), modificado (modified), y preparado (staged).

- Confirmado significa que los datos están almacenados de manera segura en tu base de datos local.
- Modificado significa que has modificado el archivo pero todavía no lo has confirmado a tu base de datos.
- Preparado significa que has marcado un archivo modificado en su versión actual para que vaya en tu próxima confirmación.

Esto nos lleva a las tres secciones principales de un proyecto de Git: el directorio de Git (Git directory), el directorio de trabajo (working directory), y el área de preparación (staging area).

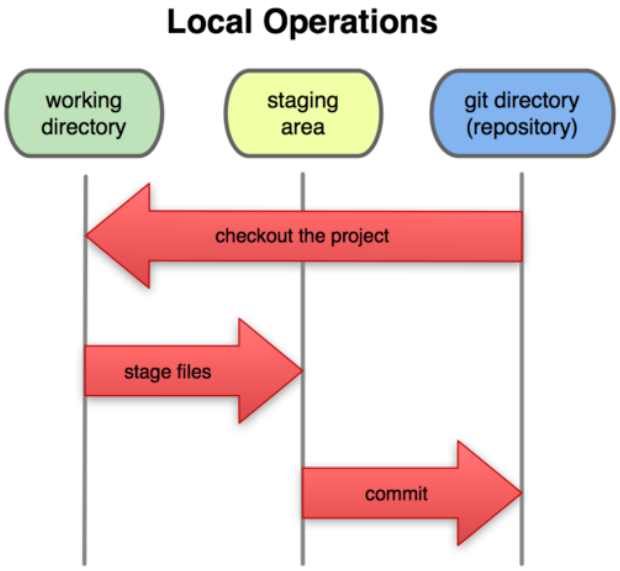

Figura 8: Directorios principales de un proyecto en Git[16]

El directorio de Git es donde Git almacena los metadatos y la base de datos de objetos para tu proyecto. Es la parte más importante de Git, y es lo que se copia cuando clonas un repositorio desde otro ordenador.

El directorio de trabajo o working directory es una copia exacta del proyecto en una de sus versiones. Estos archivos son extraídos de la base de datos de Git y se guardan en el disco local para su modificación.

La staging área es un archivo, que normalmente forma parte del directorio de Git, y que guarda la información relativa a los cambios que van a incluirse en la siguiente entrega.

Flujo de trabajo básico en Git:

Tras la modificación y edición de los archivos correspondientes del working directory. Se preparan dichos archivos, y se añaden instantáneas de todos ellos al staging área. Una vez que se han confirmado los cambios, se cogen dichos archivos de la staging área y se sube la instantánea al repositorio de Git. Si dentro del directorio de Git ya existe una versión concreta de un archivo, se acepta dicho cambio y se acepta como confirmada (committed). Si desde que se extrajo del repositorio el archivo ha sufrido cambios, pero dichos cambios se añaden a la staging área, eso implica que está preparada (staged). Y si desde que se extrajo del repositorio el archivo ha sufrido cambios, pero no se han preparado se dice que está modificada (modified).

3.3.3. Automatización de Tareas

El Universo entero tiende a la complejidad y el caos, y los proyectos de software, lejos de ser una excepción, confirman y aceleran esta tendencia. Y a medida que se amplían en alcance, complejidad o equipo, también crece el esfuerzo necesario para mantenerlos, evolucionarlos y desplegarlos tras cada nueva versión.

Parte de este esfuerzo proviene del conflicto existente entre las prioridades del proyecto en su entorno de desarrollo, donde prima la ergonomía del desarrollador y la legibilidad del código, y las características que ha de tener la aplicación ya publicada, donde esto se convierte en irrelevante y se prioriza la eficiencia y la economía de recursos.

Muchos frameworks de desarrollo web incorporan mecanismos más o menos transparentes al desarrollador para —por ejemplo— concatenar y minimizar scripts y hojas de estilos. De este modo el programador puede trabajar con componentes modulares y ficheros optimizados para la legibilidad, que son transformados al vuelo en versiones optimizadas para la velocidad de descarga o de interpretación.

Pero a medida que el proyecto gana en complejidad o en requisitos de eficiencia esta facilidad puede resultar insuficiente. Pongamos otro ejemplo: para trabajar con iconos un diseñador puede preferir trabajar con una miríada de ficheros independientes generados en lotes por su herramienta de diseño favorita y que le resultan fáciles de mantener. Y unos metros más allá su compañero desarrollador o administrador de sistemas puede ver con terror como esta práctica dispara las peticiones HTTP necesarias para servir la página, y con ello el consumo de recursos y el tiempo de transferencia.

¿Cómo resolver este conflicto? En los últimos dos o tres años están cobrando relevancia algunas herramientas dirigidas a programadores y que podríamos denominar de automatización de tareas o de compilación de proyectos (frontend build tools, developer automation tools, task runners o streaming build systems). Se trata fundamentalmente de Grunt [18] (2012), que mantiene con Gulp (2013) un interesante duelo por el favor de los desarrolladores, y GNU Make (1977. Sí: mil novecientos setenta y siete), la popular y casi omnipotente utilidad del mundo Unix.

Más allá de las diferencias conceptuales o técnicas, la idea detrás de estas herramientas es común y sencilla: automatizar la ejecución de tareas repetitivas con la finalidad de transformar un proyecto de su forma óptima para desarrollo a su forma óptima para publicación o despliegue.

Un ejemplo probablemente común en proyectos online que han adquirido una cierta complejidad podría ser un workflow consistente en todas o algunas de las siguientes tareas:

- Linting de código, particularmente JavaScript.
- Optimización de las imágenes del proyecto, recortando, comprimiendo, ajustando la calidad o realizando conversiones de formato.
- Minimización o conversión de ficheros de tipografía.
- Generación de una hoja de sprites a partir de ficheros con iconos independientes.
- Compilación de ficheros Sass, Less, CoffeScript…
- Ejecución de tests unitarios, de integración o de usuario (Selenium).
- Comprobación automática de enlaces rotos.
- Supresión de comentarios y compactación de la salida.
- Generación automática de documentación o APIs a partir del código.
- Validación semántica de lenguajes W3C.
- Concatenación y minimización de scripts u hojas de estilos.
- Vaciado (flush) de cachés.
- Despliegue automático en producción.

El coste de tener que realizar manualmente estas tareas puede ser inasumible, especialmente en proyectos con un desarrollo ágil, iteraciones frecuentes y muchas versiones y publicaciones. Esta es la dificultad que tratan de resolver las herramientas descritas. En ningún orden en particular:

Grunt, originaria del mundo de JavaScript, implementa una filosofía basada en configuración mejor que programación, y sus numerosísimos plugins facilitan las tareas más frecuentes y minimizan la curva de aprendizaje.

GNU Make, la herramienta con que nuestros abuelos compilaban sus programas Fortran en pantallas de fósforo verde, goza de excelente salud y proporciona a los usuarios

más fieles a la filosofía Unix workflows potentes y flexibles desde una herramienta estándar y casi universal.

3.4. Catálogo de requisitos

Actualmente, la web del Servicio de Deportes está dentro del dominio de la Universidad Francisco de Vitoria y esto se mantendría en un futuro. Por lo que se ha comprobado con la herramienta Pingdom, el servidor es suficientemente rápido y no necesitaremos mejoras en este aspecto.

Identificador	**REQ_FIS_001**	**Nuevo departamento**
Descripción:	Deberá existir un departamento que se encargue de la administración y gestión de la página así como de las actualizaciones de la página.	
Prioridad	Alta	
Identificador	**REQ_TEC_001**	**SO del alojamiento**
Descripción:	El alojamiento web tendrá que realizarse sobre un Sistema Operativo Linux ya que el desarrollo web se hará en el lenguaje PHP.	
Prioridad	Alta	
Identificador	**REQ_TEC_002**	**Espacio web**
Descripción:	Se deberán tener reservados al menos 250MB para el proyecto por posibles actualizaciones así como subida de imágenes	
Prioridad	Medio	
Identificador	**REQ_TEC_003**	**Base de datos**
Descripción:	Se deberá tener una base de datos MySQL con, al menos, 100MB.	

Prioridad	Alta	
Identificador	**REQ_HUM_001**	**Personal administración**
Descripción:	Deberá haber, al menos, una persona encargada de la administración del Servicio de Deportes.	
Prioridad	Necesaria	
Identificador	**REQ_HUM_002**	**Personal gestión**
Descripción:	Sería recomendable que hubiera una persona encargada de la gestión del Servicio. No es tan necesaria como la del punto anterior ya que las funciones del gestor las puede realizar también el administrador.	
Prioridad	Media	
Identificador	**REQ_INT_001**	**Adaptarse a los usuarios**
Descripción:	La aplicación web debe adaptarse a los conocimientos de los usuarios, es decir, que la persona que entre en la web pueda realizar las funciones que le ofrece ésta.	
Prioridad	Alta	
Identificador	**REQ_INT_002**	**Navegación intuitiva**
Descripción:	Tanto la página como los distintos paneles de control existentes en la misma deberán ser claros para facilitar el buen uso de las personas que lo usen.	
Prioridad	Alta	
Identificador	**REQ_INT_003**	**Interfaces específicas**
Descripción:	Se deberán realizar interfaces específicas para los administradores y gestores y, también, para los usuarios, permitiendo a cada uno de ellos realizar todas las tareas que le sean permitidas por el sistema.	
Prioridad	Recomendada	
Identificador	**REQ_INT_004**	**Elementos funcionales**

Descripción:	La página contará con formularios, cajas de texto, botones, etc. necesarios para el correcto funcionamiento de la misma.	
Prioridad	Alta	
Identificador	**REQ_RES_001**	**Integridad referencial**
Descripción:	La base de datos deberá cumplir la característica de integridad referencial teniendo todas sus tablas relacionadas entre sí.	
Prioridad	Alta	
Identificador	**REQ_FUN_001**	**Identificación**
Descripción:	Existirán dos tipos de identificación en forma de un único formulario para los usuarios y para administradores de la página.	
Prioridad	Alta	
Identificador	**REQ_FUN_002**	**Gestión de deportes**
Descripción:	Los administradores podrán crear, ver, modificar y borrar los deportes que el sistema necesite.	
Prioridad	Alta	
Identificador	**REQ_FUN_003**	**Gestión de pistas**
Descripción:	Los administradores podrán crear, ver, modificar y borrar las pistas del Servicio. Para que haya una pista, previamente deberá haber un deporte al que ésta estará asociada.	
Prioridad	Alta	
Identificador	**REQ_FUN_003**	**Gestión de reservas**
Descripción:	Los administradores y los gestores tendrán acceso a todas las reservas hechas por los usuarios. Podrán crear nuevas, verlas o cancelarlas.	

	Los usuarios, por su parte, podrán realizar nuevas reservas, cancelar las ya reservadas y verlas desde su panel de control.	
Prioridad	Alta	
Identificador	**REQ_FUN_003**	**Gestión de usuarios**
Descripción:	Los administradores y los gestores podrán ver la información general de los usuarios registrados así como borrarlos del sistema. Los usuarios, por su parte, podrán, después de darse de alta, modificar sus datos personales desde su panel de control.	
Prioridad	Alta	

Tabla 2: Catálogo de requisitos

3.5. Especificación de casos de uso

El objetivo de esta tarea es especificar cada caso de uso, desarrollando el escenario. Para completar los casos de uso será preciso especificar información relativa a:

- Descripción del escenario, es decir, cómo un actor interactúa con el sistema, y cuál es la respuesta obtenida.
- Precondiciones y poscondiciones.
- Identificación de interfaces de usuario.
- Condiciones de fallo que afectan al escenario, así como la respuesta del sistema (escenarios secundarios).

En escenarios complejos, es posible utilizar como técnica de especificación los diagramas de transición de estados, así como la división en casos de uso más simples. En nuestro caso ha sido suficiente con dividir los casos de uso generales en unos más simples o específicos.

Para la obtención de esta información es imprescindible la participación activa de los usuarios.

Caso de Uso 0. Servicio de Deportes

En este apartado se van a ver los casos de uso del nivel más general y los primeros con los que deben encontrarse los actores del sistema los cuales explicamos brevemente a continuación. Los actores que están involucrados en este caso de uso son:

- Administrador: usuario que tiene acceso a la gestión de todos los casos de uso presentados en el diagrama que se observa a continuación.
- Usuario: sólo tendrá acceso reducido a la gestión de reservas.

Según el caso general que comentábamos anteriormente, este sería el caso de uso más general de la aplicación web:

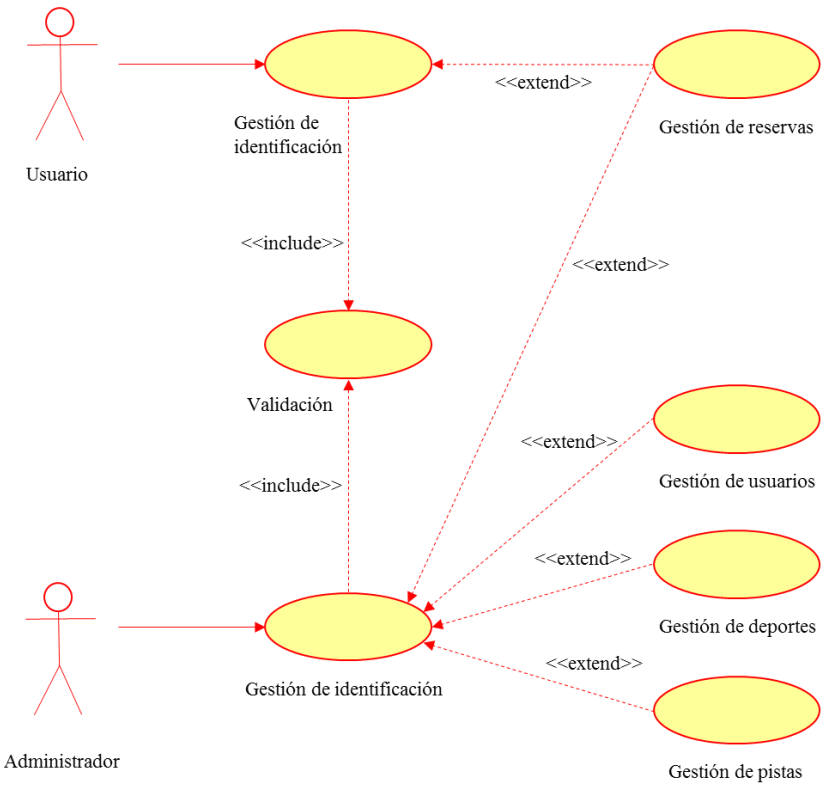

Diagrama 16: Casos de Uso 0 – Servicio de Deportes

Como se observa, se ha generalizado desde el administrador hasta llegar al usuario que es el más básico de los actores que intervienen en el Servicio de Deportes.

Cada uno de los actores, en primer lugar tendrá que pasar por la gestión de identificación que no es más que el formulario de acceso a sus respectivos paneles de controles. Cuando realizan la validación, si es correcta, se les otorga unos permisos u otros dependiendo del tipo de actor que sea.

El administrador hereda las funciones del usuario en primera instancia. Esto quiere decir que las funciones que puede desempeñar el usuario también pueden ser realizadas por el administrador pero no al contrario.

Identificador	**CU_001**	
Fecha Creación / Revisión	06/05/2015	06/05/2015
Versión / Autor	2.1	Ignacio López
Nombre:	Gestión de Identificación.	
Descripción	Gestiona el proceso de identificación o login de los usuarios en la web o de los administradores en el panel de contol.	
Precondiciones		
Poscondiciones	Si el proceso de validación se lleva a cabo de manera correcta los usuarios podrán pasar a los casos de uso	
Condiciones de fallo		

Tabla 3: Caso de uso de gestión de identificación

Identificador	**CU_002**	
Fecha Creación / Revisión	06/05/2015	06/05/2015
Versión / Autor	2.0	Ignacio López

Nombre:	Gestión de Deportes.
Descripción	Gestiona los deportes que se van a poder practicar
Precondiciones	El código del deporte o el nombre del mismo no pueden estar almacenados en la base de datos en el momento de insertar uno nuevo.
Poscondiciones	
Condiciones de fallo	

Tabla 4: Caso de uso de gestión de deportes

Identificador	**CU_003**	
Fecha Creación / Revisión	06/05/2015	06/05/2015
Versión / Autor	2.0	Ignacio López
Nombre:	Gestión de Pistas.	
Descripción	Gestiona las pistas donde se van a realizar los deportes que previamente se han creados.	
Precondiciones	No puede haber dos pistas del mismo deporte con el mismo identificador.	
Poscondiciones		
Condiciones de fallo		

Tabla 5: Caso de uso de gestión de pistas

Identificador	**CU_004**	
Fecha Creación / Revisión	06/05/2015	06/05/2015
Versión / Autor	2.0	Ignacio López
Nombre:	Gestión de Usuarios.	

Descripción	Gestiona los distintos usuarios que se han registrado
Precondiciones	No puede haber dos usuarios con el mismo DNI, correo electrónico o número de expediente.
Poscondiciones	
Condiciones de fallo	Si en el momento del registro existe algún usuario con el mismo DNI, correo electrónico o número de expediente se mostrará un mensaje de error al usuario para que modifique los campos.

Tabla 6: Caso de uso de gestión de usuarios

Identificador	**CU_005**	
Fecha Creación / Revisión	06/05/2015	06/05/2015
Versión / Autor	2.0	Ignacio López
Nombre:	Gestión de Reservas.	
Descripción	Gestiona las reservas que existen en la base de datos y las que se realizarán en el futuro.	
Precondiciones	Ser administrador o si se trata de un usuario deberá haber iniciado sesión como tal.	
Poscondiciones		
Condiciones de fallo	Si no se está registrado como administrador o, si es usuario, no se ha iniciado sesión será imposible que pueda ver reserva alguna. Si en alguno de los pasos se introduce algún valor incorrecto también tendremos un mensaje de error por pantalla.	

Tabla 7: Caso de uso de gestión de reservas

Caso de Uso 1. Gestión de Deportes

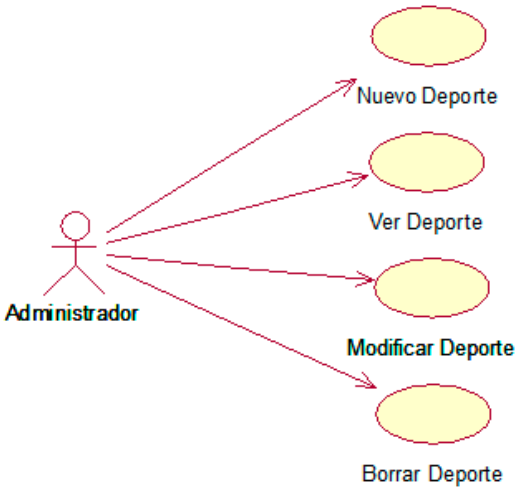

Diagrama 17: Casos de uso 1 – Gestión de Deportes

El caso de uso que se representa sobre estas líneas es el de la gestión de deportes. A éste, sólo se tiene acceso cuando tras la gestión de identificación se han obtenido los privilegios de administrador.

El administrador, podrá crear nuevos deportes así como ver los ya creados, es decir, los que en el momento de realizar esta función estén guardados en la base de datos. También podrá modificar los deportes ya creados, por ejemplo, para cambiar el nombre o el precio de los mismos.

Por último, el administrador tendrá la posibilidad de borrar alguno de los deportes que se hayan creado previamente con lo que desaparecería automáticamente todas las pistas vinculadas a dicho deporte y todas las reservas vinculadas a esas pistas y deporte.

Identificador	**CU_001.01**	
Fecha Creación / Revisión	07/05/2015	07/05/2015
Versión / Autor	2.0	Ignacio López

Nombre:	Nuevo Deporte
Descripción	El administrador podrá crear los deportes.
Precondiciones	Haber iniciado sesión como administrador.
Poscondiciones	
Condiciones de fallo	

Tabla 8: Caso de uso para crear un nuevo deporte

Identificador	**CU_001.02**	
Fecha Creación / Revisión	07/05/2015	07/05/2015
Versión / Autor	2.0	Ignacio López
Nombre:	Ver Deporte	
Descripción	El administrador podrá ver la información general de los deportes creados.	
Precondiciones	Haber iniciado sesión como administrador.	
Poscondiciones		
Condiciones de fallo		

Tabla 9: Caso de uso para ver un deporte

Identificador	**CU_001.03**	
Fecha Creación / Revisión	07/05/2015	07/05/2015
Versión / Autor	2.0	Ignacio López
Nombre:	Modificar Deporte	
Descripción	El administrador podrá modificar los datos de alguno de los deportes creados.	

Precondiciones	Haber iniciado sesión como administrador.
Poscondiciones	
Condiciones de fallo	

Tabla 10: Caso de uso para modificar un deporte

Identificador	**CU_001.04**	
Fecha Creación / Revisión	07/05/2015	07/05/2015
Versión / Autor	2.0	Ignacio López
Nombre:	Borrar Deporte	
Descripción	El administrador podrá borrar alguno de los deportes creados.	
Precondiciones	Para poder borrar un deporte éste deberá estar creado cuando se vaya a realizar dicho borrado. Haber iniciado sesión como administrador.	
Poscondiciones		
Condiciones de fallo		

Tabla 11: Caso de uso para borrar un deporte

Caso de Uso 2 Gestión de Pistas

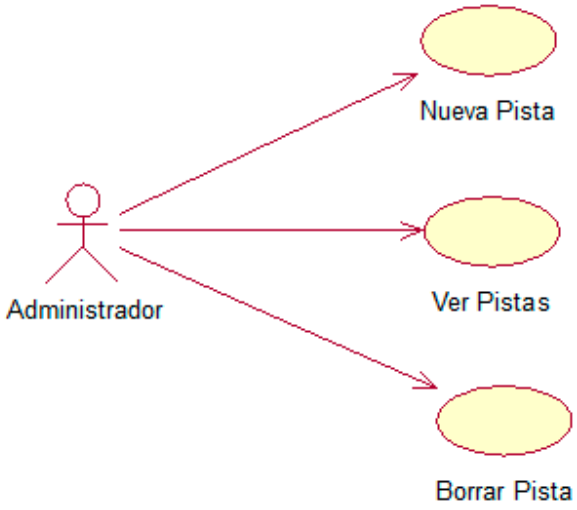

Diagrama 18: Casos de uso 2 – Gestión de Pistas

Este caso de uso es el de la gestión de pistas. Al igual que en el caso de uso 1, a él sólo tiene acceso el actor que haya obtenido los privilegios de administrador al entrar en el panel de administración.

A diferencia del anterior, el administrador sólo tiene tres opciones: crear una nueva pista, ver las pistas disponibles y borrar alguna de éstas.

Cuando se crea una pista se pide el deporte al cual pertenece, es decir, el deporte que se va a practicar en dicha pista. Dependiendo de ese deporte y el número de pista se le asignará un identificador a ésta.

También se tendrá la posibilidad de ver todas las pistas creadas y, finalmente, borrar alguna de ellas.

Identificador	**CU_002.01**	
Fecha Creación / Revisión	06/05/2015	06/05/2015

Versión / Autor	2.0	Ignacio López
Nombre:	Nueva Pista	
Descripción	El administrador podrá crear la pista de un deporte determinado.	
Precondiciones	El deporte que se practica en esa pista ha de estar creado previamente. Haber iniciado sesión como administrador.	
Poscondiciones		
Condiciones de fallo		

Tabla 12: Caso de uso para crear una nueva pista

Identificador	**CU_002.02**	
Fecha Creación / Revisión	06/05/2015	06/05/2015
Versión / Autor	2.0	Ignacio López
Nombre:	Ver Pistas	
Descripción	Se podrán visualizar las pistas que están almacenadas en la base de datos.	
Precondiciones	Haber iniciado sesión como administrador.	
Poscondiciones		
Condiciones de fallo		

Tabla 13: Caso de uso para ver una pista

Identificador	**CU_002.03**	
Fecha Creación / Revisión	06/05/2015	06/05/2015
Versión / Autor	2.0	Ignacio López

Nombre:	Borrar Pista
Descripción	El administrador podrá borrar alguna de las pistas que están almacenadas en la base de datos.
Precondiciones	Haber iniciado sesión como administrador. Que la pista exista previamente para poder ser borrada.
Poscondiciones	
Condiciones de fallo	

Tabla 14: Caso de uso para borrar una pista

Caso de Uso 3. Gestión de Usuarios

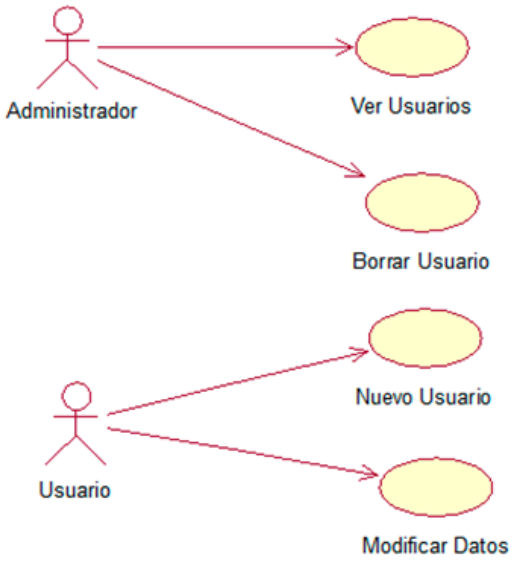

Diagrama 19: Casos de uso 3 – Gestión de Usuarios

La gestión de usuarios como tal la tiene el administrador. Hemos incluido al usuario porque en cierto modo actúa en esta gestión aunque sólo sería una gestión de usuario asimismo ya que sólo tiene las funciones de darse de alta o registrarse y de modificar sus datos personales.

Por su parte, el actor que entre con los privilegios de administrador tendrá la posibilidad de ver los usuarios en forma de listado y de borrar o modificar los usuarios que quieran con lo que serían actualizados/eliminados de nuestra base de datos.

Identificador	**CU_003.01**	
Fecha Creación / Revisión	06/05/2015	06/05/2015
Versión / Autor	2.0	Ignacio López
Nombre:	Ver Usuarios.	
Descripción	Se visualizarán todos los usuarios que se hayan registrado en la página.	
Precondiciones	Haber iniciado sesión como administrador.	
Poscondiciones		
Condiciones de fallo		

Tabla 15: Caso de uso para ver los usuarios

Identificador	**CU_003.02**	
Fecha Creación / Revisión	06/05/2015	06/05/2015
Versión / Autor	2.0	Ignacio López
Nombre:	Borrar Usuario.	
Descripción	Se podrán borrar usuarios que estén registrados en nuestra base de datos.	
Precondiciones	Haber iniciado sesión como administrador.	
Poscondiciones		
Condiciones de fallo		

Tabla 16: Caso de uso para borrar un usuario

Identificador	CU_003.03	
Fecha Creación / Revisión	06/05/2015	06/05/2015
Versión / Autor	2.0	Ignacio López
Nombre:	Nuevo Usuario.	
Descripción	El usuario que entre en la página podrá registrarse en la misma.	
Precondiciones		
Poscondiciones		
Condiciones de fallo	Se notificará cuando un dato introducido no sea correcto. Si existe algún usuario con el mismo DNI, correo electrónico o número de expediente se mostrará un mensaje de error al usuario para que modifique campos.	

Tabla 17: Caso de uso para crear un nuevo usuario

Identificador	CU_003.04	06/05/2015
Fecha Creación / Revisión	06/05/2015	06/05/2015
Versión / Autor	2.0	Ignacio López
Nombre:	Modificar Datos.	
Descripción	El usuario podrá modificar sus datos personales.	
Precondiciones	Haber iniciado, el propio usuario, una sesión.	
Poscondiciones		
Condiciones de fallo	Si alguno de los datos introducidos es erróneo se le notificará. Si existe algún usuario con el mismo DNI, correo electrónico o número de expediente se mostrará un mensaje de error al usuario para que modifique campos.	

Tabla 18: Caso de uso para modificar los datos de usuario

Caso de Uso 4. Gestión de Reservas

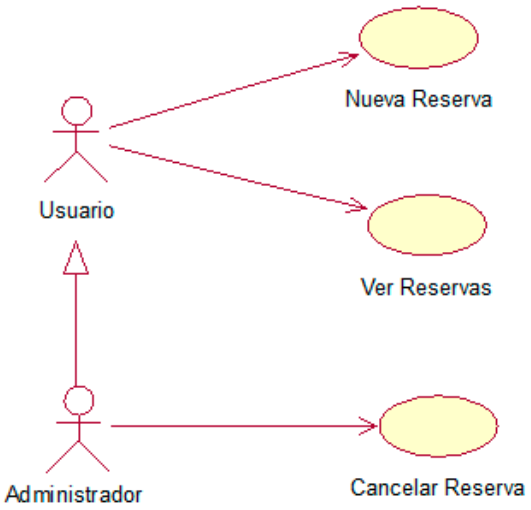

Diagrama 20: Casos de uso 4 – Gestión de Reservas

Por último, se tiene la gestión de reservas cuyo diagrama de caso de uso se puede ver en el anterior diagrama.

Como se observa, el administrador tendrá las mismas funciones que el usuario con la única salvedad de que podrá cancelar una reserva realizada por cualquier usuario.

Cada uno de los actores podrá realizar nuevas reservas.

Identificador	**CU_004.01**	
Fecha Creación / Revisión	06/05/2015	06/05/2015
Versión / Autor	2.0	Ignacio López
Nombre:	Nueva Reserva.	
Descripción	Cualquiera de los roles (administrador o usuario) podrá realizar una nueva reserva.	
Precondiciones	Se deberá haber iniciado sesión con alguno de los roles previamente dichos.	

Poscondiciones	
Condiciones de fallo	Si no hay pistas disponibles en el horario deseado, el usuario no podrá confirmar la reserva Si un usuario no inicia una sesión, no podrá confirmar la reserva.

Tabla 19: Caso de uso para crear una nueva reserva

Identificador	**CU_004.02**	
Fecha Creación / Revisión	06/05/2015	06/05/2015
Versión / Autor	2.0	Ignacio López
Nombre:	Ver Reservas.	
Descripción	Se visualizará la información de la o las reservas registradas en la base de datos.	
Precondiciones	Se deberá haber iniciado sesión como administrador o usuario.	
Poscondiciones		
Condiciones de fallo		

Tabla 20: Caso de uso para ver una reserva

Identificador	**CU_004.03**	
Fecha Creación / Revisión	06/05/2015	06/05/2015
Versión / Autor	2.0	Ignacio López
Nombre:	Cancelar Reserva.	
Descripción	Cancelación de una reserva previamente realizada por alguno de los actores del sistema.	
Precondiciones	Haber iniciado sesión como administrador.	

Poscondiciones
Condiciones de fallo

Tabla 21: Caso de uso para cancelar una reserva

3.6. Diagramas de actividad

A continuación, se muestran los diagramas de actividad que muestran con más exactitud el ciclo de vida, por llamarlo de alguna manera, de las funciones principales del proyecto.

3.6.1. Gestión de deportes

3.6.1.1. Nuevo deporte

Diagrama 21: Diagrama de actividad para nuevo deporte

En primer lugar el administrador tendrá que acceder a s perfil, una vez que accede a la sección del Menú principal Gestión de Pistas, podrá rellenar un formulario con el nuevo deporte. Se almacenará dicho deporte en la base de datos y, finalmente, se le

mostrará al administrador un mensaje informándole que el deporte se ha creado de manera satisfactoria.

3.6.1.2. Ver deportes

Diagrama 22: Diagrama de actividad para ver los deportes

Cuando el administrador entra en la página de gestión de pistas, automáticamente se realiza una consulta para ver todos los deportes que están registrados en la base de datos y sus pistas asociadas.

Una vez la base de datos extraiga la información de cada uno de los deportes se devuelve rellenando una tabla para mostrar al administrador la información más destacada de cada uno de ellos.

3.6.1.3. Modificar deportes

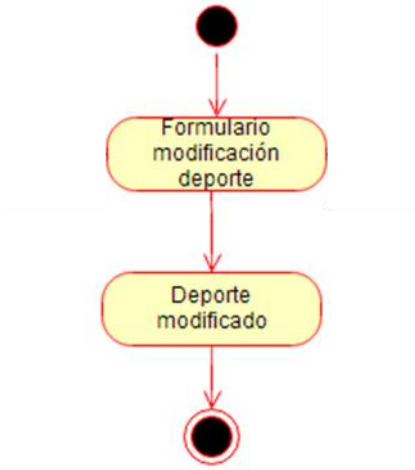

Diagrama 23: Diagrama de actividad para modificar un deporte

Desde la página de gestión de pistas, al hacer clic en el nombre de un deporte se activará dicho convirtiéndose en editable, al modificarlo se detectará el cambio y se guardará en base de datos.

3.6.1.4. Borrar deportes

Diagrama 24: Diagrama de actividad para eliminar un deporte

Desde la página de gestión de pistas, al hacer clic en el icono de la papelera en cualquier deporte, se lanzará una sentencia SQL, que hará un Delete on cascade de la base de datos, es por eso, que se ha incluido un cuadro de diálogo JavaScript indicando si está seguro de querer borrar el deporte y toda la información relacionada. En caso de confirmar el borrado se eliminará el deporte, sus pistas asociadas y las reservas vinculadas.

3.6.2. Gestión de pistas

3.6.2.1. Nueva pista

Diagrama 25: Diagrama de actividad para crear una nueva pista

El usuario deberá rellenar un formulario en el que tiene que indicar el nombre de la pista, el precio del alquiler, precio de la luz y el deporte que se va a practicar (se cargan los creados en la base de datos).

Una vez enviado se inserta en la base de datos y, finalmente, se mostrará al administrador un mensaje con la confirmación de la creación de dicha pista.

3.6.2.2. Ver pistas

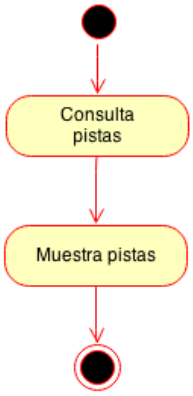

Diagrama 26: Diagrama de actividad para ver las pistas

Cuando el administrador entra en esta página, automáticamente se realiza una consulta para ver todas las pistas que existen en ese momento en la base de datos.

Una vez extraída la información se devuelve para que se le muestre en una tabla con la información más importante de cada una de ellas.

3.6.2.3. Borrar pista

Diagrama 27: Diagrama de actividad para borrar una pista

Cuando el administrador entra en esta página, automáticamente se realiza una consulta para mostrar todas las pistas existentes. Cuando seleccionemos una de ellas y hagamos clic en el icono papelera la pista será eliminada.

Finalmente, aparecerá por pantalla un mensaje para confirmarle al administrador que se ha borrado de manera satisfactoria.

3.6.3. Gestión de reservas

3.6.3.1. Nueva reserva

Diagrama 28: Diagrama de actividad para crear una nueva reserva

En primer lugar, el usuario accede al deporte que desea practicar y se le mostrarán las pistas asociadas a dicho deporte, en un primer momento la vista mostrará el día actual. Si el usuario elige otro día, la página mostrará la disponibilidad de esos días. En el caso del usuario podrá ver tres tipos de estados de pistas: Ocupada, Reservada por él o libre. El usuario sólo podrá interactuar con las reservadas por él para cancelarlas o con las libres para reservarlas.

Finalmente, una vez insertada de manera correcta la reserva se mostrará un mensaje al usuario para informarle de que la reserva se ha creado satisfactoriamente.

3.6.3.2. Ver reserva

Cuando entramos en este apartado automáticamente hacemos la consulta a la base de datos buscando cualquier reserva que esté registrada con el id del usuario que ha iniciado sesión.

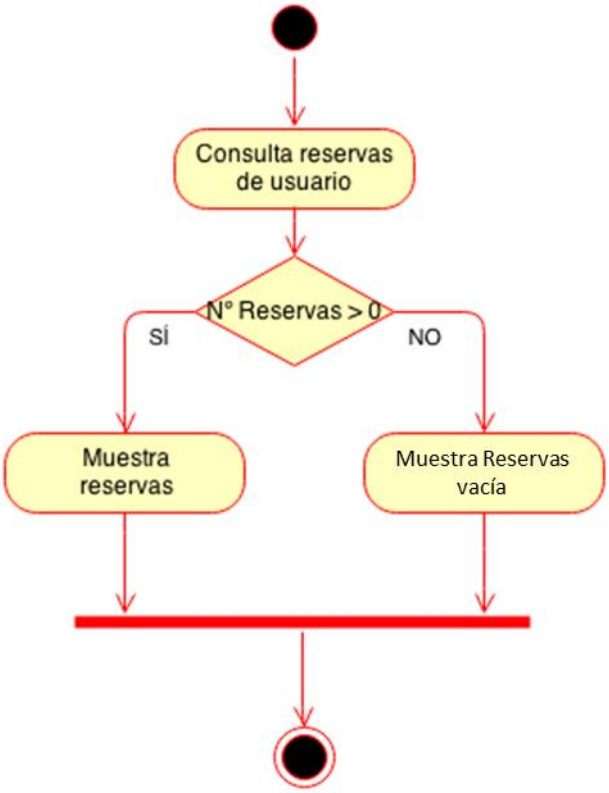

Diagrama 29: Diagrama de actividad para ver una reserva

Tendremos, entonces, dos opciones. La primera es que no haya ninguna reserva con el id que le hemos pasado a la base de datos. Si se da este caso se mostraría la pantalla de Reservas pero vacía.

Por el contrario, si hubiera al menos una reserva, se mostrará una tabla en la que podremos ver la información más relevante de las reservas que tengamos almacenadas.

3.6.3.4. Cancelar reserva

Cuando el administrador accede a la página de Reservas le aparecerá un icono de Papelera para poder borrar las reservas que desee. Del mismo modo el usuario final, podrá actuar análogamente con sus propias reservas.

Diagrama 30: Diagrama de actividad para la cancelación de una reserva

3.6.4. Gestión de usuarios

3.6.4.1. Nuevo usuario

En primer lugar, el usuario tendrá que rellenar un formulario con distintos campos. Algunos de estos serán obligatorios a rellenar por el usuario y otros opcionales. Si a la hora de enviar dicho formulario alguno de los campos obligatorios no están rellenados no permite realizar dicho envío.

Si, por el contrario, todos los campos obligatorios han sido rellenos se pasa a comprobar si el número de expediente, el DNI o el correo electrónico introducido coinciden con el de alguno de los usuarios ya registrados. Si es así volvemos al formulario anterior para que el usuario los modifique y vuelva a validarlo.

Si todo es correcto almacenamos sus datos en la base de datos con lo que quedaría registrado dicho usuario en la aplicación. Finalmente, se mostrará un mensaje al mismo informándole de que ha sido dado de alta correctamente en el Servicio.

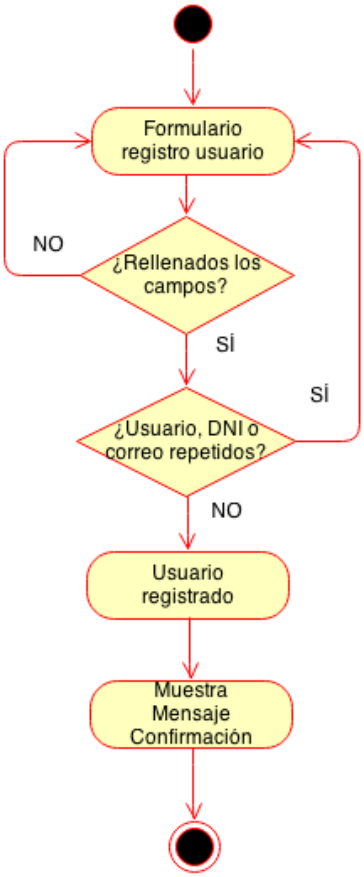

Diagrama 31: Diagrama de actividad para registrarse un usuario

3.6.4.2. Ver usuarios

El administrador al entrar tendrá un listado con todos los usuarios. Si se quiere ver un usuario en concreto se puede filtrar por cualquiera de los campos de la tabla que se muestra.

Diagrama 32: Diagrama de actividad para ver usuarios

La segunda de las opciones es la de mostrar todos los usuarios. El administrador no tendrá que rellenar ningún formulario sino que cuando entra en la página se realiza una consulta para extraer todos y cada uno de los usuarios existentes en la base de datos.

Una vez extraída la información de cada uno de ellos se mostrará completando una tabla cuyas columnas se corresponderán con la información más destacada de cada uno de los usuarios que en ese momento estén dados de alta en el Servicio.

3.6.4.3. Modificar datos

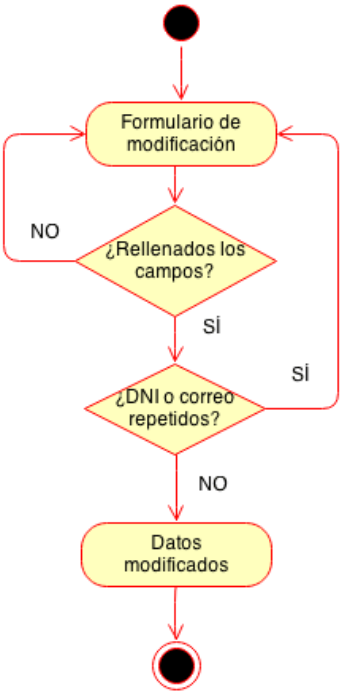

Diagrama 33: Diagrama de actividad para modificar datos personales

En primer lugar, se mostrará un formulario muy parecido al de registro en el que los campos vendrán rellenados con los datos actuales del usuario, el único campo que no se mostrará será la contraseña. Cuando quiera modificar alguno de ellos sólo tendrá que posicionarse en el campo que desee y cambiarlo. Antes de enviar el formulario se comprobará que todos los campos están rellenados, si no, se les indicará que deben hacerlo para poder enviarlo.

Cuando envíe dicho formulario, se validarán los datos. No podrá coincidir ni el DNI ni el email con el de cualquiera de los usuarios que ya existen en la base de datos. Si no se cumplen estos requisitos se le volverá a mandar al formulario. En caso contrario, es decir, si todo es correcto, actualizaremos el usuario con la nueva información en la base de datos y finalizaremos la modificación del mismo.

3.6.4.4. Borrar usuario

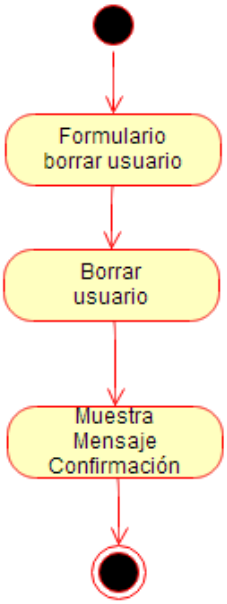

Diagrama 34: Diagrama de actividad para borrar un usuario

Esta función se encarga de eliminar de la base de datos a un usuario previamente registrado. El administrador localizará al usuario dentro del listado de usuarios y pulsando la papelera enviará el evento para que se realice el borrado de la base de datos.

Finalmente, se mostrará al actor que realice dicha función un mensaje informándole que la eliminación del usuario se ha realizado correctamente.

3.6.5. Gestión de identificación

3.6.5.1. Inicio de sesión

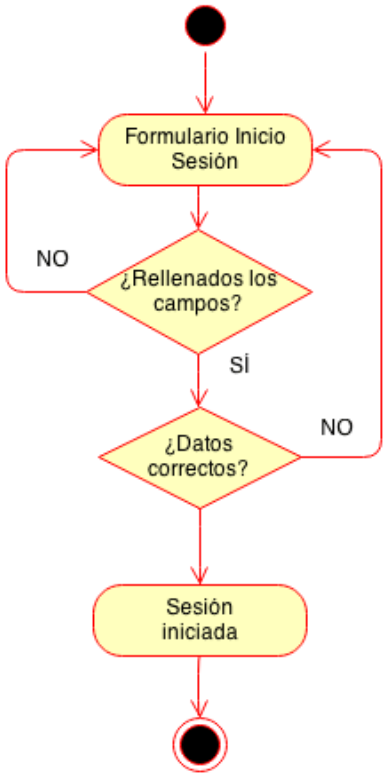

Diagrama 35: Diagrama de actividad de inicio de sesión

El usuario o administrador (en sus diferentes paneles) tendrán que rellenar un formulario con los únicos campos de correo electrónico y contraseña.

Si alguno de los campos no está rellenado o no es válido se le indica que no puede continuar con el intento de inicio de sesión. Si, por el contrario, ambos campos han sido rellenados se pasa a validar los datos introducidos ya dentro de la base de datos.

Si el correo electrónico y la contraseña se corresponden, el usuario tendrá iniciada la sesión desde ese momento. Para informarle de ello, se le enviará a la página de inicio y se mostrará por pantalla un mensaje confirmando que el inicio de sesión ha sido correcto.

3.7. Diagramas de navegabilidad

3.7.1. Administrador

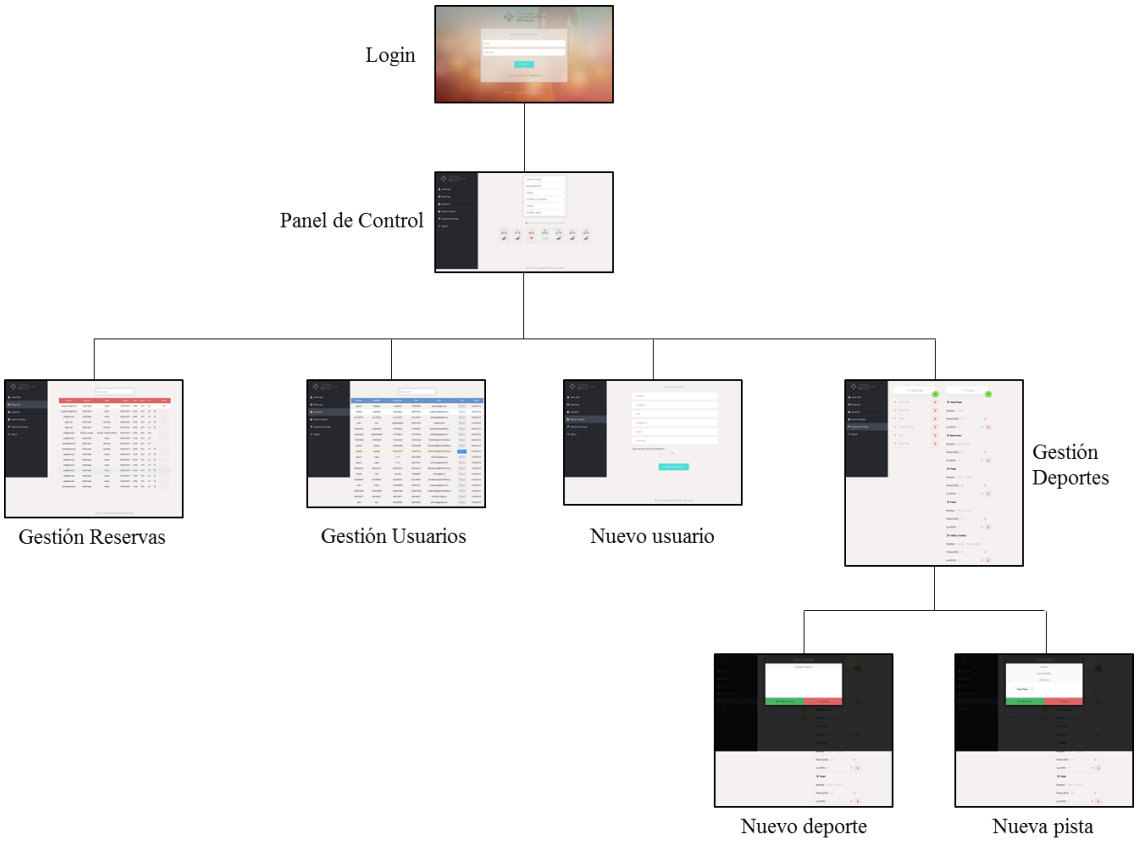

Diagrama 36: Diagrama de navegabilidad del administrador

3.7.2. Usuario

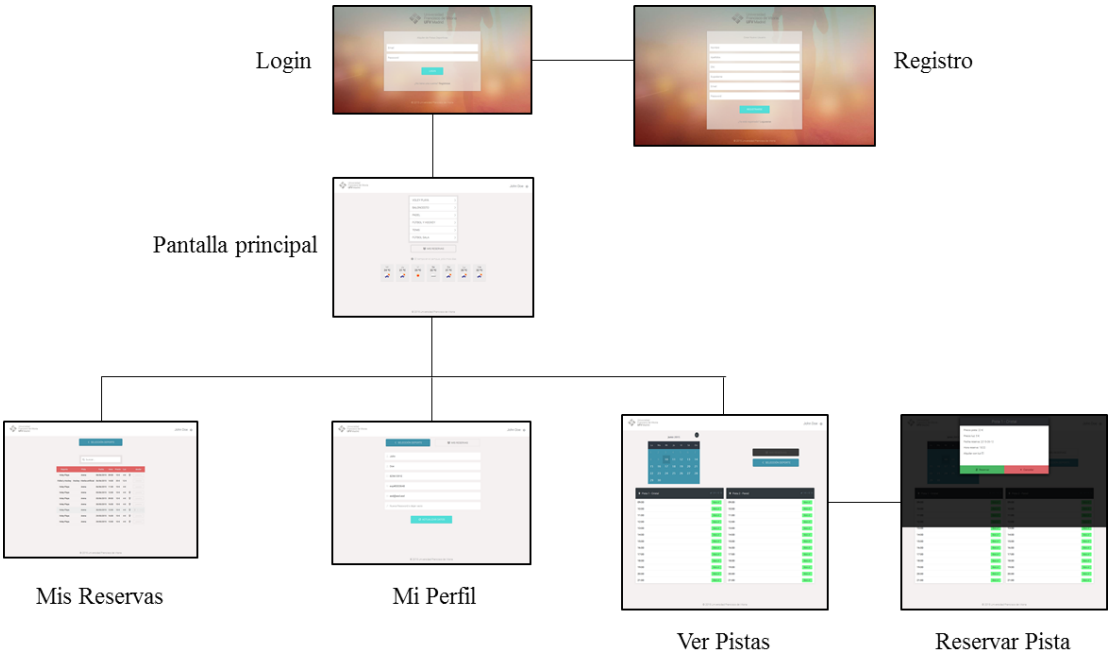

Diagrama 37: Diagrama de navegabilidad del usuario

3.8. Definición de los servicios desarrollados

En el proyecto se identifican dos actores:

- Usuarios
- Administradores

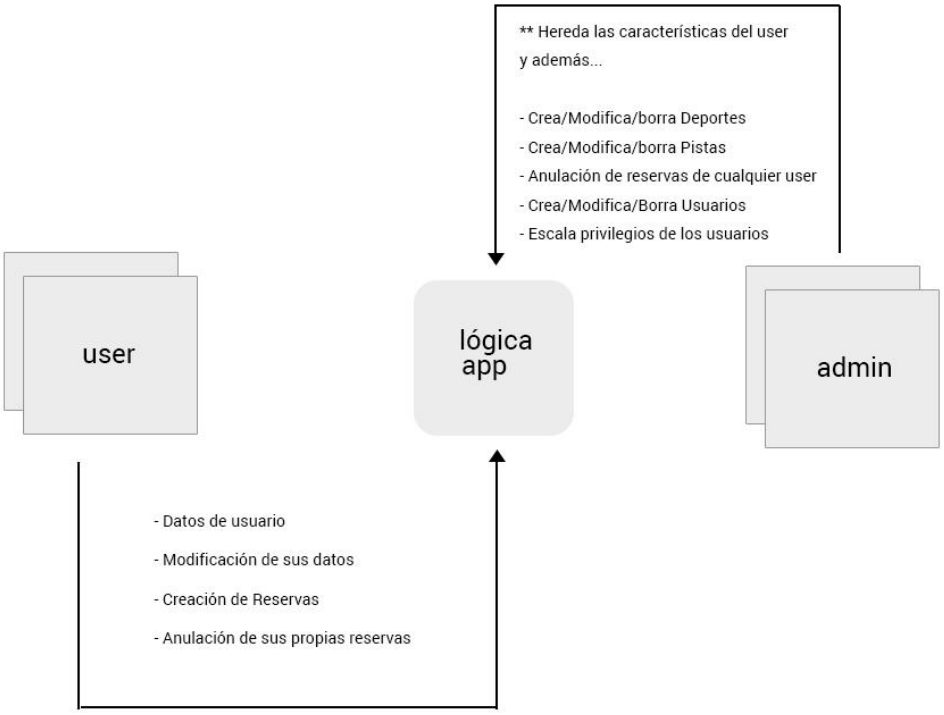

Figura 9: Modelo relación roles de usuario

Servicios comunes para los roles de administrador y usuario:

A la hora de definir los servicios, hay que tener en cuenta que muchos de ellos tienen una visión y funcionalidad específica según el rol del usuario (Administrador/Usuario), por lo que detallaremos esas especificaciones propias de cada rol al explicar cada uno de los módulos en los casos en los que aplique.

a) Alta de usuarios - src/js/views/registro.js

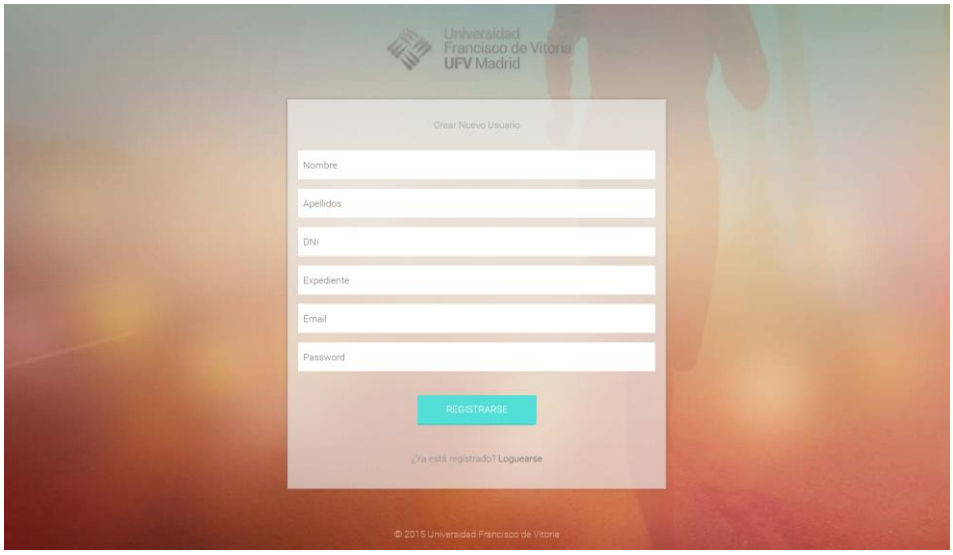

Figura 10: Pantalla de Registro

A este módulo se accede desde la pantalla de Login. Su único fin es registrar alumnos. Los Administradores sólo pueden ser registrados por otro Administrador desde el panel de control. Se le solicitan al usuario sus datos:

- Nombre
- Apellidos
- Mail
- Expediente
- Password

Antes de enviar los datos a la api pasan por una función de validación que comprueba que los campos tengan la longitud adecuada, que el formato del DNI encaje con un DNI válido y que el email sea de formato válido bajo una expresión regular que lo comprueba.

Si falla la validación se corta la ejecución y se muestra un aviso de qué campos están incorrectos y por qué.

Si la validación es OK se envía al servicio de la API: nuevoUsuario/ que espera los campos: nombre, apellidos, expediente, dni, password y mail.

NOTA: La contraseña se envía siempre cifrada en Sha1, nunca un gestor de la aplicación ni con acceso a las bases de datos sabrá cuál es la contraseña de cada usuario, se guardan cifradas en la base de datos.

La API rest comprueba que no exista otro usuario con mismo email, dni o expediente. Si hay coincidencias devuelve un error indicando que ese usuario ya existe. Si se registra correctamente devuelve un success con un mensaje de "usuario creado correctamente".

La aplicación capturará el success y usará un bus de eventos para lanzar el evento altaDeUsuarioOK que acepta un parámetro por el que se envía el correo electrónico del usuario. Al detectar la Aplicación dicho evento redirecciona a la pantalla de Login y deja rellenado el email del cliente para que solo falte introducir la password.

b) Login - src/js/views/login.js

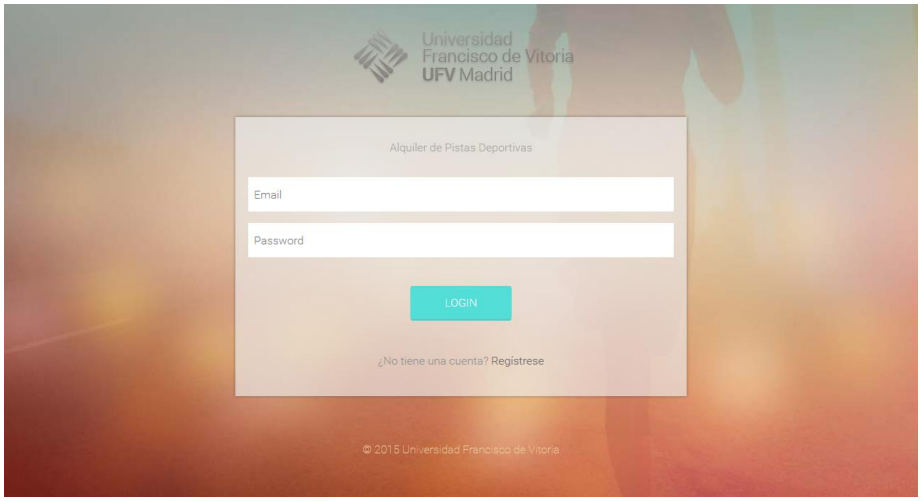

Figura 11: Pantalla Login

Al acceder a la aplicación se comprueba si el usuario que accede está logueado. En caso contrario se le redirige a la pantalla de Login. Esta pantalla consta de 2 inputs para poder introducir las credenciales. Desde aquí se puede pivotar hacia Alta de Usuarios por si un usuario es nuevo y carece de cuenta, haciendo clic en "regístrese"

Al introducir las credenciales se consulta mediante el servicio de la api rest login/, quien se encargará de comprobar si los datos son veraces. En caso afirmativo devuelve un success y retorna un objeto JSON con los datos del Usuario: rol, dni, expediente, nombre, apellidos y email. Si por el contrario falla indica que alguno de los datos está equivocado sin decir cuál para evitar que descubran por fuerza bruta qué emails están ya registrados en la base de datos.

Si el resultado de la consulta a la api de login es OK, se le redirige a la parte privada interna de la aplicación.

c) Persistencia de Sesión - src/js/models/sesion.js

A partir de ahora la aplicación coge dos caminos diferentes dependiendo del rol del usuario que se ha logueado.

- Usuario: rol = 0
- Admin: rol = 1

Se guardará un objeto usuario que contendrá los siguientes datos: Nombre, Apellidos, Email, Expediente y Rol.

Este objeto usará un patrón singleton de instancia única con métodos privados de tal forma que sólo la aplicación es capaz de crearlo, esto nos garantiza la integridad de la seguridad. Estará disponible el siguiente método para lectura para poder comprobar en cualquier punto qué rol tiene el usuario logueado, para poder diferenciar qué mostrar o qué no.

```
islogged: function() {
```

```
    var response = 'unlogged';
    var sesion = Sesion.getInstance();
    var sesionRol = sesion.get('rol');
    if(Number(sesionRol) === 1) response = 'admin';
    else if(Number(sesionRol) === 0) response = 'user';
    return response;
  },
```

d) Loading...

Este módulo se encarga de mostrar u ocultar una animación css3 con dos circunferencias girando sobre sí mismas que impide tocar ningún botón de la pantalla. Se activa cuando comienza una petición Ajax y termina al recibir la respuesta, que después manejará la app.

Se encuentra localizado dentro del router de la aplicación.

```
loader: function () {
    var loading = $('#loading');
    var documento = $(document);
    var modalCalendario = $('#modalCalendario');
    documento.ajaxStart(function () {
      loading.show(0);
    });
    documento.ajaxComplete(function () {
      loading.fadeOut(500);
    });
  },
```

e) Menú Principal - src/js/views/header.js

Usuario: En el caso de usuario el menú consta de logotipo principal que siempre redirige a la HOME. En la parte derecha se muestra el nombre del usuario recogido del objeto de sesión. Si se pulsa nos lleva a la ficha modificar usuario. Por motivos de usabilidad si la pantalla es estrecha como en móviles se cambia mediante css el nombre del usuario por un icono tipo user con la misma función de llevarnos hasta la ficha de usuario. Existe un tercer botón que desloguea y borra la instancia de la sesión, impidiendo el acceso hasta que no se vuelva a loguear otro usuario.

Admin: En admin el diseño es notablemente distinto ya que además lleva una serie de links a las distintas partes de gestión. Este menú también es responsive ocultándose en pantallas pequeñas, el administrador tendrá que pulsar sobre el botón del menú para desplegarlo.

f) SignOut/cerrar sesión - Este es un servicio que desloguea al usuario. Destruye el objeto de sesión y redirige al login.

```
logout: function (event) {
    event.preventDefault();
    Sesion.destroySesion();
    var sesion = Sesion.getInstance();
    Backbone.app.navigate("/login", { trigger: true });
  },
```

g) Listado de deportes - src/js/views/deporte-single.js src/js/views/deportes-lists.js

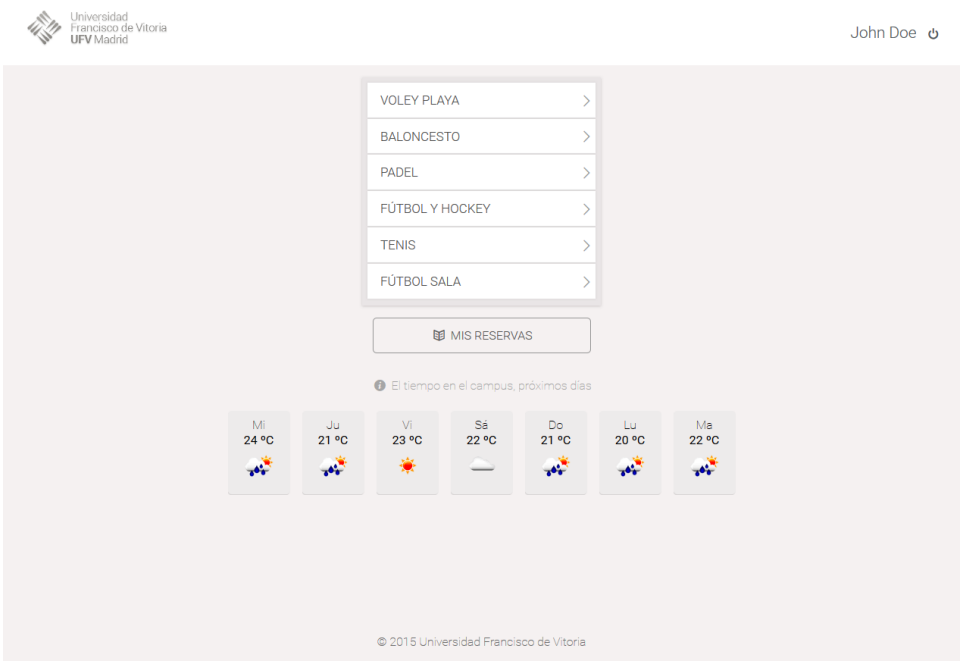

Figura 12: Pantalla Listado Deportes

Esta vista aparece como bienvenida nada más entrar a la aplicación. Consulta a la API pistas/ que devuelve un objeto JSON con los deportes que contienen pistas. Si se pulsa sobre un deporte se accede a las pistas asociadas a dicho deporte.

h) Previsión de El tiempo próximos 7 días:

src/js/views/eltiempo.js

Para este módulo se ha utilizado una api de un tercero que nos da la información meteorológica de los próximos días

API EL TIEMPO

Al consultarla nos devuelve un JSON con los datos medios por día y la url del icono a mostrar por cada día. Lo recogemos y mediante una iteración montamos el un bucle en el frontend que nos lo muestre como se puede apreciar en la vista. Al hacer la consulta a la API se le mandan las coordenadas del campus de la UFV:

```
getTiempo: function(){
    var self = this;
    var lat = 40.439854;
    var lon = -3.834995;
    $.getJSON(API_WEATHER_WEEK_URL + "lat=" + lat + "&lon=" + lon +
"&cnt=7&units=metric&mode=json",
      function getCurrentWeather(data){
        // console.log(data);
        var cityWeather = {};
        // cityWeather.zone = data.name;
        // cityWeather.icon = IMG_WATHER_URL + data.weather[0].icon + ".png";
        // cityWeather.temp = data.main.temp - 272.15;
        // cityWeather.temp = cityWeather.temp.toFixed(0);
        cityWeather.zone = data.city.name;
        cityWeather.days = [];
        for(var i=0; i<data.list.length; i++){
          cityWeather.days[i] = {
                temp : data.list[i].temp.day.toFixed(0),
                icon: IMG_WATHER_URL + data.list[i].weather[0].icon + ".png",
                date: Moment.unix(data.list[i].dt).locale("es").format('dd')
          };
        }
        // console.log(cityWeather);
        //render
        self.render(cityWeather);
      });
  },
```

Este módulo solo tiene carácter informativo, no aporta nada a la lógica del proyecto.

i) Listado de pistas - src/js/views/deporte-single.js src/js/views/deportes-lists.js

Al listado de pistas se accede desde el listado de deportes. A esta vista se le pasa el deporte sobre el que ha pulsado el usuario y se envía al servicio pistas/

Este servicio espera dos campos:

- id_deporte
- Día

Se le pasa el día de hoy siempre salvo al interaccionar con el calendario y cambiar de fecha que se le pasa entonces la fecha seleccionada.

Este servicio devuelve un objeto Json con las pistas, las horas de cada pista y su estado, si está libre u ocupado. A partir de aquí hay dos variantes por rol:

Usuario: Si es un usuario estándar se comprueba el id de usuario de cada reserva con el del usuario logueado, si estos dos son iguales se entiende que la reserva es propietaria, y se puede anular, si son distintos ids se entiende que la reserva es de un tercero y aparece como "no anulable". Si no tiene reserva asociada se mostrará como libre y podrá reservarse.

Admin: En el caso del administrador no se hace la comprobación de quién ha reservado la pista ya que tiene los privilegios para anular cualquier reserva por lo cual sólo ve dos estados, libre o reservado con capacidad para anularlo.

El Administrador también puede reservar pistas.

j) Reservar - src/js/views/reservar.js

Para reservar hay que pulsar sobre una hora que contenga la etiqueta "libre". Se abre un modal de confirmación que nos pregunta si queremos alquilar con luz. Al confirmar la reserva se envían los datos de qué usuario la crea, si quiere o no luz al

servicio de la api: reserva/ con los siguiente campos: id_usuario, id_pista, id_hora, fecha_pista, luz, anulado

La api hace una segunda comprobación de que no exista dicha reserva, en caso de existir devuelve un error avisando que ya está reservada y que por favor refresques el navegador. Esto evita la concurrencia de reservas en caso de que dos usuarios independientes reserven al mismo tiempo la misma pista con misma fecha y hora.

Si la reserva no existe la crea y devuelve un success con un mensaje de "reserva hecha correctamente" que se mostrará al usuario. En este punto lanza un evento por el bus de eventos para que se refresque la vista (sin refrescarse el navegador, vía ajax). Al refrescarse la vista vuelve a pedir al servicio pistas\ la información para mostrarla de forma actualizada y cambiará el estado de la pista que estaba libre a reservada.

k) Anular - src/js/views/anular.js

Para anular hay que pulsar sobre una hora que contenga la etiqueta "anular reserva". Se abre un modal de confirmación que nos pregunta si queremos anularla. Al confirmar la anulación se envía el id_reserva al servicio de la api: anularReserva/ con los el siguiente campo: id_reserva.

Si la respuesta es un success, se procede igual que en el punto anterior, se lanza un evento por el bus de eventos para que se refresque la vista. Al refrescarse la vista vuelve a pedir al servicio pistas\ la información para mostrarla de forma actualizada y cambiará el estado de la pista que estaba reservada a libre. Ahora cualquier usuario verá libre esta hora en esta pista este día y podrá reservarla.

l) Calendario - src/js/views/calendario.js

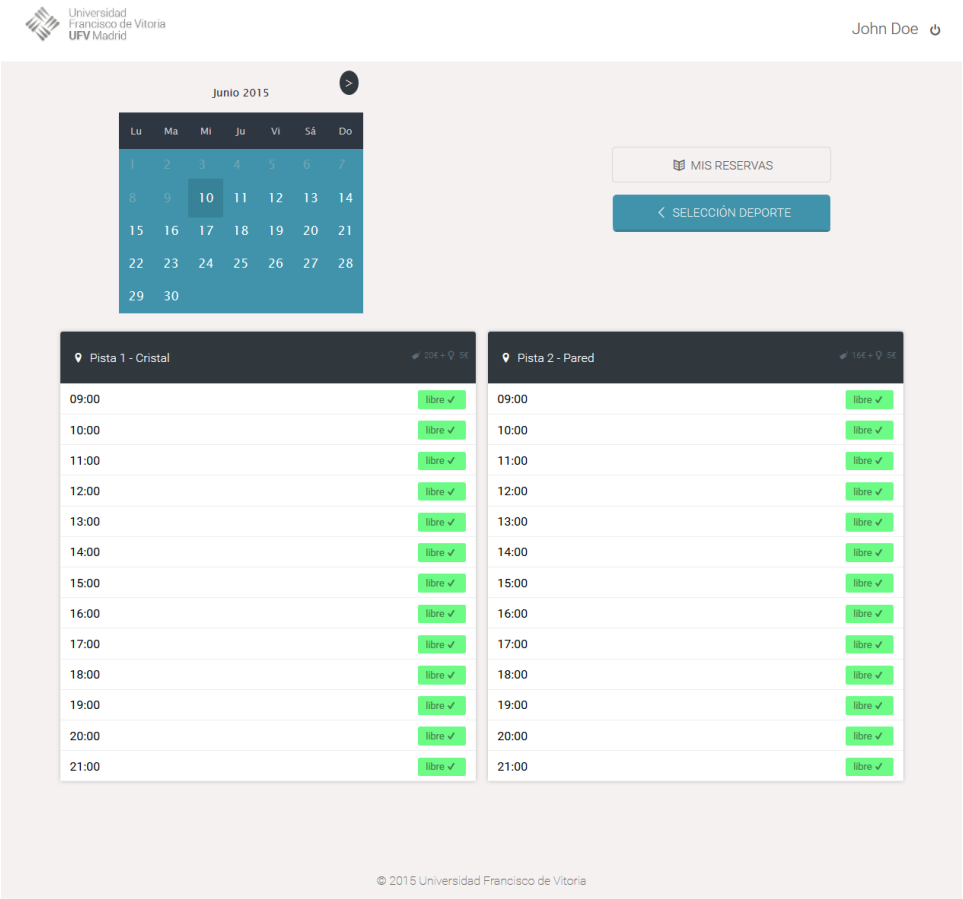

Figura 13: Pantalla Selección de pista y Calendario

Este módulo genera un calendario usando el plugin Jquery UI. Si no se le pasa fecha mostrará la de hoy, si se le pasa una fecha por la url la coge y muestra dicho día.

```
getDateUrl: function(){
  var f = window.location.href;
  f = f.split("/");
  f = _.last(f).trim();
  isF = Moment(f, 'YYYY-MM-DD', true).isValid();
  if (isF === false) f = Moment().format('YYYY-MM-DD');
  return f;
```

},

Al cambiar una fecha se vuelve a llamar al método pistas para refrescar la vista y mostrar las reservas asociadas de ese día.

m) Ficha/modificación de usuario- src/js/views/perfil.js

Permite listar los datos del usuario que son recogidos del objeto sesión. El usuario tiene la capacidad de modificar cualquiera de los datos siempre teniendo en cuenta la previa validación de los mismos.

Aunque el usuario esté logueado por seguridad se le pide la clave actual otra vez antes de lanzar el servicio de actualización.

Tras pasar la validación en el front se envían los datos a la api actualizarUsuario/. La api comprueba que nuevamente si cambia mail, expediente o DNI que no coincidan con los de otro usuario.

Si los cambios han sido efectivos se responde con un success y se le muestra el mensaje al cliente: “datos de usuario actualizados”. Se lanza un evento para actualizar la vista con los nuevos datos. En caso de que todos los datos enviados fueran iguales se le avisará al usuario que no se ha modificado ningún dato ya que todos los enviados eran los mismos que los que se encuentran en la base de datos.

n) Listado de reservas - src/js/views/reservas-list.js

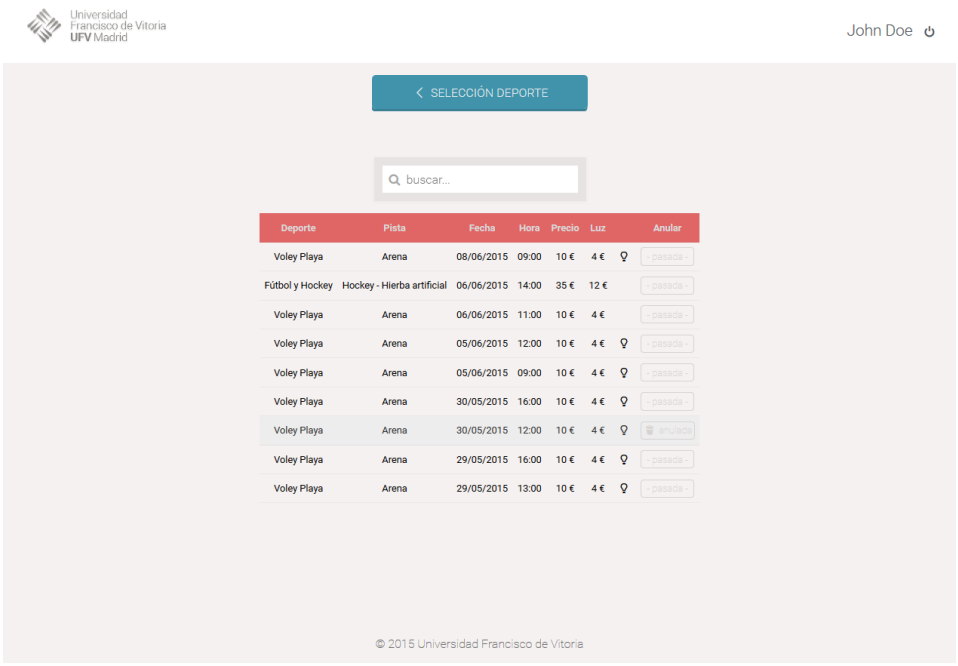

Figura 14: Pantalla listado Reservas con perfil usuario

Usuario: Al acceder a esta sección se consulta el servicio de la api reservasUsuario/ que espera como parámetro el id_usuario.

Nos devuelve un objeto JSON con todas las reservas de dicho usuario y su estado.

En el frontend comprobamos si las reservas no anuladas han pasado ya de fecha en cuyo caso le adjuntamos la etiqueta "-pasada-". Para esto usamos una función helper que comprueba si la fecha es anterior a mañana.

```
models = models.map(function (model) {
        if(model.fecha_pista){
           datePista = Moment(model.fecha_pista).unix();
           model.fecha_pista = Moment(model.fecha_pista).format('DD/MM/YYYY');
        }
        compareDate = Moment().unix();
```

```
        //console.log(datePista);
        if( ((datePista + (24*60*60) ) - compareDate) < 0){
            if(Number(model.anulado) === 0) model.anulado = 2;
        }
        return model;
    });
```

El resto de las reservas mostrarán anulado o reservado.

Cuando una reserva no está anulada ni pasada se puede anular pulsando sobre el icono de la papelera roja, este botón llama a la api 'anular/' y se le pasa el id de la reserva. En el success de la anulación se lanza un evento para refrescar la vista.

Admin: El administrador tiene ligeras diferencias.

- Muestra las reservas de todos los usuarios
- Puede anular las reservas de cualquier usuario

Para esto en caso de ser rol == 1 se consultará el servicio reservasAdmin/ que listará las reservas de todos los usuarios.

o) Buscador de reservas

En el frontend las reservas son guardadas dentro de una colección de modelos, cada modelo sería una fila. Esto nos permite diferenciar cada uno de los campos del modelo que son las columnas por las que podemos filtrar de forma instantánea sin tener que pedir de nuevo el servicio para cada búsqueda ya que tenemos guardado de la petición anterior todas las reservas completas.

La búsqueda sucede al teclear. Comienza en el evento change del input buscando coincidencias en cualquier columna con el string introducido y filtrando el contenido.

filter: function() {

```
            var what = this.get('what').trim().toLowerCase(),
                where = this.get('where'),
                lookin = (where==='all') ? ['nombre_deporte', 'nombre_pista', 'fecha_pista',
'inicio', 'precio_pista', 'precio_luz', 'anulado','luz'] : where,
                models;
            if (what==='') {
                models = this.collection.models;
            } else {
                models = this.collection.filter(function(model) {
                    return _.some(_.values(model.pick(lookin)), function(value) {
                        return ~value.toLowerCase().indexOf(what);
                    });
                });
            }
            this.filtered.reset(models);
    },
```

Listados de servicios particulares del rol Administrador:

p) Alta nuevos Usuarios - src/js/views/registro-admin.js

Esta sección permite al Administrador crear usuarios/administradores. Funciona igual que el registro público en cuanto a lógica y validación.

Si se selcciona el checkbox "administrador" se mandarán los datos al sercivio nuevoAdmin/ por el contrario si se deja sin seleccionar se mandarán los datos a al servicio nuevoUsuario/ de la API REST.

```
var rolAdmin = $('#rolAdmin').is(':checked'),
        url = (rolAdmin)? '/api/nuevoAdmin' : '/api/nuevoUsuario',
```

q) Listado de usuarios - src/js/views/users-list.js

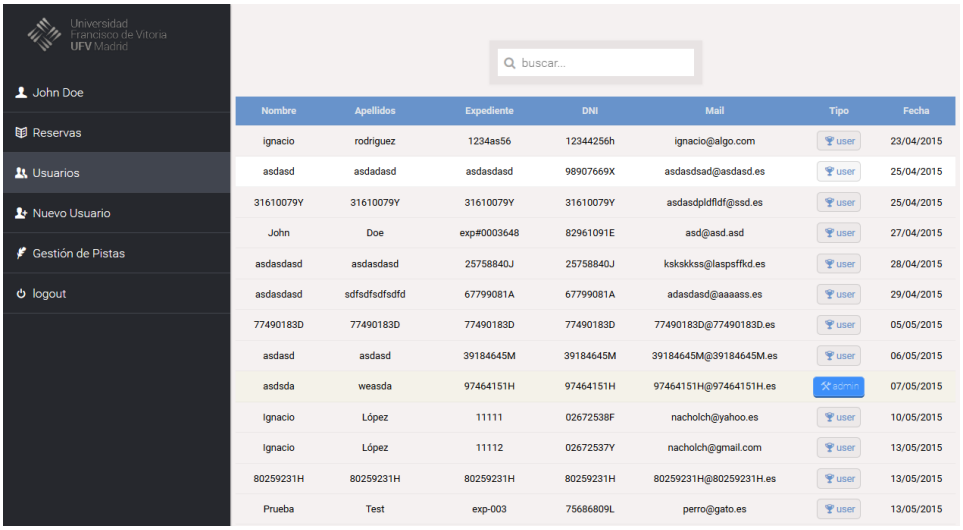

Figura 15: Listado usuarios desde perfil administrador

Al acceder a este apartado la aplicación pide a la API REST el servicio usuarios/ vía GET sin pasar ningún parámetro. Ésta devuelve un objeto JSON con el conjunto de usuarios de la aplicación incluido su rol.

Si se pulsa sobre uno de los usuarios se crea una vista nueva new userPerfil cargando los datos del usuario seleccionado para poder modificarlos o borrarlos. Se puede cambiar también el rol de usuario/admin escalando o degradando sus permisos de acceso.

```
events: {
    'click': 'goFichaUser'
  },
  goFichaUser: function(event){
    if(window.userperfil!=undefined) {
      window.userperfil.resetear();
      window.userperfil.undelegateEvents();
    }
    this.userPerfil = new UserPerfil({ model: this.model });
```

```
        this.userPerfil.render();
        window.userperfil = this.userPerfil;
        $('html, body').animate({ scrollTop: 0 }, 'slow');
    }
```

Para eliminar un usuario se le pasa al API rest el id_usuario al servicio eliminarUsuario/

Para modificar los datos de usuario se mandan los datos al servicio actualizarUsuarioAdmin/

r) Buscador de usuarios

Este servicio es el mismo que el buscador de reservas solo que aplicado a la colección de usuarios.

En este caso se utiliza también un helper para en el front distinguir de manera visual si en el listado es un usuario sin privilegios o un administrador.

s) Gestión de Pistas

Este módulo es el gestor principal de las pistas de la aplicación. Permite a los administradores dar de alta nuevos deportes, nuevas pistas asociadas a un deporte y asignarles precio.

También permite la modificación de cualquier campo así como la eliminación de pistas y deportes.

También se pueden eliminar tanto pistas como deportes.

CREAR: Para crear un nuevo deporte/pista hay que pulsar sobre el botón + de color verde. Esto dispara una modal con una serie de inputs de datos a rellenar.

En caso de ser nuevo deporte pedirá el nombre del deporte.

Si es una nueva pista pedirá el nombre de la pista, los precios de la pista y de la luz y un combo para seleccionar entre uno de los deportes existentes previamente creados para asignarlas a dicho deporte.

Los servicios para dar de alta son para las pistas nuevaPista/ y para los deportes nuevoDeporte/

MODIFICAR: Para modificar se ha de dar doble click sobre el campo que se desee actualizar. Al dar doble click transformamos el dato en un input, si se cambia el dato la aplicación detectará el evento change del input y lanzará el servicio contra la API REST modificarPista/ ó modificarDeporte.

BORRAR: Si se elimina un deporte, en base de datos se hará un DELETE ON CASCADE con las pistas asociadas y también sobre las reservas asociadas a dichas pistas.

Para eliminar un deporte/pista hay que pulsar sobre la papelera roja y se envía al servicio de la API eliminarDeporte/ ó eliminarPista/ pasándole el ID de la pista o del deporte según corresponda.

Ante un success de cualquiera de los tres eventos anteriores (CREAR/MODIFICAR/BORRAR) se lanza un evento para que se refresque la vista y mostrar los datos actualizados de la base de datos.

4. Resultados obtenidos

En primer lugar, se ha obtenido una aplicación web que se podrá colocar directamente en un servidor. Lógicamente, para que funcione tendremos que crear una base de datos en el propio servidor y cargarla con la estructura que hemos decidido sobre su desarrollo. Para realizar este proceso, explicaremos en el manual de instalación cómo realizar dichos pasos para que sea más entendible por parte de la persona que se haga

cargo del proyecto. Esta aplicación, permite a los usuarios revisar los deportes disponibles en la Universidad, así como las pistas asociadas a los mismos y la disponibilidad de las mismas. Por otro lado, la UFV podrá gestionar de manera online la gestión de pistas y el alquiler de las mismas.

En segundo lugar, hemos obtenido esta memoria en la que se resume la realización y explicación del proyecto. Para ello, se parte desde el punto en el que se encuentra la página del Servicio de Deportes de la Universidad Francisco de Vitoria, se especifican las características que tendría la nueva aplicación dentro del proyecto ya existente y se termina con las mejoras que podría tener el mismo.

Y, por último, se ha realizado una explicación esquemática del proyecto donde se explica de manera gráfica y sencilla las peculiaridades del proyecto.

5. Conclusiones

El proyecto que se ha realizado ha contribuido a identificar y resaltar los puntos que hay que cubrir y considerar para llevar a cabo una buena implementación de los sistemas de información.

Partiendo de la página que, a día de hoy, se nos ofrece por parte del Servicio de Deportes y viendo las necesidades de éste se ha ido completando punto a punto hasta conseguir tener un proyecto acorde con la importancia del Servicio en una Universidad como la UFV en la que practican deporte alumnos y/o trabajadores de la misma y personas ajenas a esta.

Como objetivo principal, se ha buscado desde el principio facilitar a todos los actores que participan en este Servicio la gestión de las reservas y el resultado es el que se ha visto a lo largo de la memoria y ampliado a un rediseño de la página web que la UFV tiene en la actualidad ampliando su intuitividad y facilidad de navegación desde todos los dispositivos ya sean de escritorio o móviles.

Por último, se puede decir que lo que ofrece este proyecto es necesario y muy útil para un Servicio de Deportes que tanta gente usa y que carece, a día de hoy, de las facilidades que da éste también a la parte de administración y gestión aprovechando las nuevas tecnologías disponibles.

6. Trabajos futuros

Todos conocemos como avanzan las nuevas tecnologías y lo útiles que pueden llegar a sernos en la vida cotidiana. Un altísimo porcentaje de usuarios poseen un Smartphone que está activo continuamente y, por ello, sería conveniente crear una aplicación que facilitara aún más si cabe la gestión de reservas a los usuarios e, incluso, crear una aplicación para los administradores y gestores que también hicieran más fácil la administración del Servicio.

En un siguiente paso, convendría antes de entrar en la parcela económica, incluir una comunicación entre el sistema y el usuario, de forma que tras reserva se lance un email de confirmación que pueda servir como justificante a la hora de acceder al servicio.

Además, de cara al usuario, a la hora de hacer reservas sería mucho más sencillo poder gestionar el pago directamente desde la propia plataforma, sin tener que abonar cantidades a través del personal de la UFV. De esta forma se facilita la reserva y gestión de las instalaciones cerrando el ciclo del producto, para realizar esto lo más conveniente es integrar una pasarela de pago de la entidad bancaria vinculada con UFV. A esto habría que añadir la posibilidad de gestionar los pagos y la opción de imprimir una reserva ejecutada.

Para el mejor uso de la gestión de reservas, se debería hacer una función que analizara las reservas existentes y a partir de ahí comprobara cuándo entrará alguna reserva en que pista sería más óptimo colocarla ya que en la aplicación se obtiene una lista con las posibles pistas y se selecciona una de ellas aleatoriamente.

A parte, una mejora de la seguridad tanto en la parte de administración como en la parte de usuarios sería más que recomendable. Se ha realizado un proceso para mantener la seguridad en cuanto a contraseñas y mantenimiento de la integridad de la aplicación pero no para salvar a la aplicación web de ataques externos.

Como el propósito es dejar esta web como futura página integrada dentro del Servicio de Deportes, sería recomendable automatizar el tema de noticias y contenido de la página. Para ello, sería recomendable ampliar las secciones de la administración con, por ejemplo, gestión de noticias que serán publicadas en la propia página web.

7. Bibliografía

[1] Salazar, Lucho (2013, July). Scrum Official Guides [Online]. Available: http://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-ES.pdf

[2] Cerrada, José A (2000). Introducción a la ingeniería de software (1ª Ed.), Centro de Estudios Ramón Areces.

[3] MySQL™ Reference Manual (2015, June). MySQL™ Reference Manual [Online]. Available: http://downloads.mysql.com/docs/refman-5.5-en.a4.pdf

[4] MySQL 5.7 Reference Manual, (2015, June). MySQL™ Reference Manual [Online] http://dev.mysql.com/doc/refman/5.7/en/index.html

[5] PHP: Mysql – Manual, (2012) [Online]. Available: http://www.php.net//manual/es/book.mysql.php

[6] Oracle (2015, June), MySQL PHP API, MySQL, http://dev.mysql.com/doc/apis-php/en

[7] ¿Qué es PHP?, PHP: Hypertext Preprocessor, (2012) [Online]. Available: http://www.php.net/manual/es/intro-whatis.php

[8] Backbone JS (2015, June) [Online]. Available: http://backbonejs.org/

[9] Underscore JS (2015, April) [Online]. Available: http://underscorejs.org/

[10] Handlebars JS (2015) [Online]. Available: http://handlebarsjs.com/

[11] Moment JS (2015) [Online]. Available: http://momentjs.com/

[12] ¿Qué es PHP? PHP: Hypertext Preprocessor, (2012) [Online]. Available: http://php.net/manual/es/function.sha1.php

[13] Backbone LocalStorage JS (2015) [Online]. Available: http://backbonejs.org/docs/backbone.localStorage.html

[14] jQuery Documentation (2015) [Online]. Available: http://api.jquery.com/

[15] jQuery UI Manual (2015) [Online]. Available: https://jqueryui.com/

[16] Git Manual (2015) [Online]. Available: https://git-scm.com/doc

[17] Node.js v0.12.4 Manual & Documentation, (2015) [Online]. Available: https://nodejs.org/api

[18] Grunt.js Manual & Documentation, (2015) [Online]. Available: http://gruntjs.com/getting-started

[19] XAMPP Apache + MySQL + PHP + Perl, (2015), [Online]. Available: https://www.apachefriends.org/es/index.html

8. Anexos

8.1. Tutorial de instalación y uso

En este punto vamos a describir paso por paso cómo armar el proyecto en local desde cero para poder desarrollar y cómo se crean deploys a producción:

8.1.1. Lamp stack

Para este proyecto necesitamos tener instalado en nuestro sistema un entorno LAMP. LAMP es el acrónimo usado para describir un sistema de infraestructura de internet que usa las siguientes herramientas:

- Linux, el sistema operativo; En algunos casos también se refiere a LDAP.
- Apache, el servidor web;
- MySQL/MariaDB, el gestor de bases de datos;
- Perl, PHP, o Python, los lenguajes de programación.

La combinación de estas tecnologías es usada principalmente para definir la infraestructura de un servidor web, utilizando un paradigma de programación para el desarrollo. Existen distintas maneras de lograrlo, se puede instalar de forma nativa en el sistema o bien utilizar alguna herramienta como XAMPP [19].

XAMPP es un software libre, que hace las veces de servidor independiente de, básicamente trabaja con bases de datos del tipo MySQL, también con un servidor tipo Apache y PHP y PERL como intérpretes de lenguaje de script. El acrónimo XAMP significa que X (es válido para diversos sistemas operativos) y el resto son sus funcionalidades usables, esto es, Apache, MySQL, PHP, Perl.

Para usar XAMPP es necesario descargar el programa y establecer algunas configuraciones pertinentes para su uso con el servidor Web. XAMPP es uan herramienta

que se somete a constantes actualizaciones, tanto de ella misma como de sus componente Apache/MySQL/PHP y Perl. Para instalar XAMPP, lo primero es descargarlo.

- XAMPP: Descargar XAMPP

NOTA: Dar al archivo botón derecho y ejecutar con permisos de administrador.

Seguir los pasos de instalación y siempre verificar que se instala tanto PHP como MYSQL, el resto de opciones son opcionales.

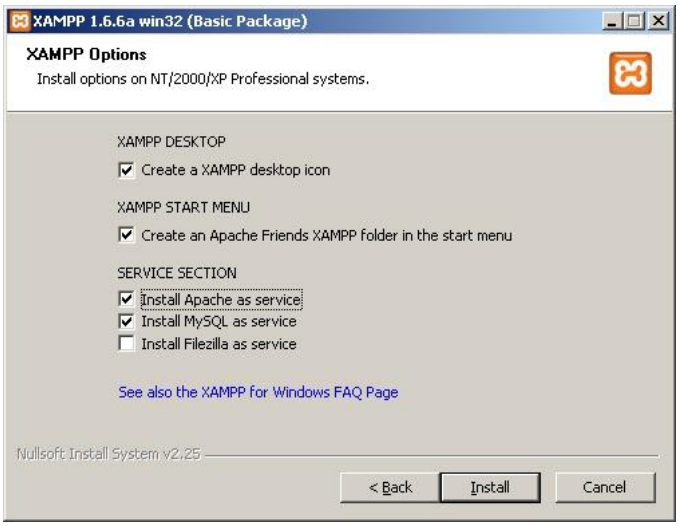

Figura 16: Instalación de XAMPP

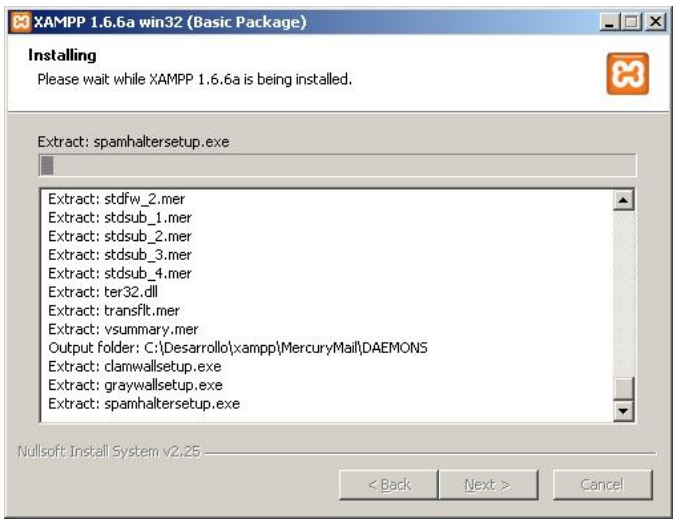

Figura 17: Instalación de XAMPP (continuación)

Al terminar la instalación arrancar el programa de nuevo en modo ejecutar con permisos de administrador y seleccionar los módulos y arrancar los servicios de PHP y MySQL

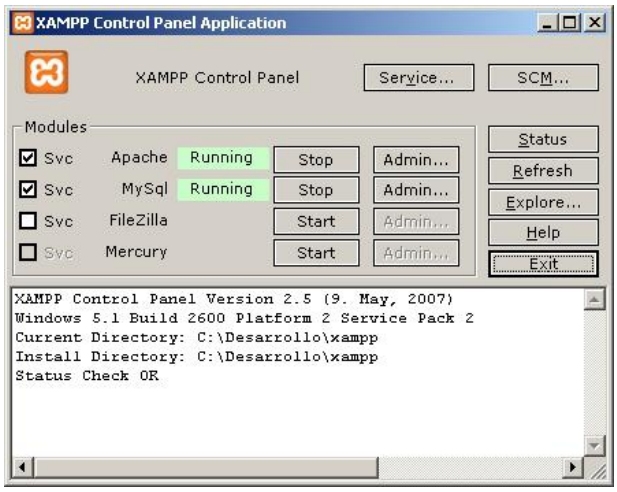

Figura 18: Panel de control XAMPP

Ahora ya tenemos instalado un servidor web y phpmyadmin para poder acceder a base de datos de una manera sencilla.

8.1.2. Git

Para instalar el proyecto msysGit sólo tenemos que descargar el instalador exe y seguir uno sencillo proceso para su instalación. Así pues desde la página de GitHub se descarga dicho archivo y se ejecuta:

- GIT: Descargar GIT

Tras la instalación, nos encontramos con la consola de línea de comandos (esto incluye un cliente SSH) y junto a ella una interfaz gráfica de usuario estándar.

Nota para el uso en Windows: Es conveniente utilizar Git con la consola que msysGit provee por defecto, que es una consola tipo Unix, en caso de necesitar usar la consola que provee por defecto Windows, la consola de línea de comandos, se ha de poner especial atención en determinados comandos puesto que en lugar de comillas simples se

usan comillas dobles (para parámetros que contienen espacios) y se debe poner entre comillas los parámetros finalizando éstos con el acento circunflejo (^) si ocupan el final de la línea.

8.1.3. Nodejs

Para poder utilizar todas las herramientas de automatización necesitamos tener instalado el motor V8 de Chrome que nos brinda Node para poder ejecutar Javascript del lado del servidor, es un intérprete de comandos gracias al cual no necesitamos estar dentro de un navegador para poder ejecutar Javascript.

Si no tenemos instalado todavía Node.JS el proceso es bastante fácil. Por supuesto, todo comienza por dirigirse a la página de inicio de NodeJS:

- Node.js: Descargar NODE

Allí encontraremos el botón para instalarlo "Install" que pulsamos y simplemente seguimos las instrucciones.

Una vez está instalado, el modo de trabajo con NodeJS es independiente de la plataforma y teóricamente no existe una preferencia dada por uno u otro sistema, Windows, Linux, Mac, etc. Sin embargo, dependiendo de nuestro sistema operativo sí puede haber unos módulos diferentes que otros, esto es, unos pueden funcionar en Linux y no así en otros sistemas, y viceversa.

Ya tenemos instalado todo lo necesario para trabajar con el proyecto ahora pasamos a configurar el entorno, obtener el código y ejecutarlo.

8.1.4. Configurar un Dominio Local

Antes de empezar definamos qué es un virtual host (también llamado host virtual): consiste en poder alojar múltiples dominios en una sola máquina.

¿En dónde son utilizados? Son utilizados en ambientes de desarrollos, por lo que solo funcionan de manera local.

Paso 1:

Lo primero que debemos hacer es crear el directorio donde alojaremos nuestros virtual hosts que contendrá el código, por ejemplo en la carpeta principal de documentos de xammp

C:/xampp/htdocs/pfc

Dentro de esta carpeta es donde guardaremos nuestros proyectos.

Paso 2:

Lo siguiente que debemos hacer es dirigirnos a C:\WINDOWS\system32\drivers\etc\ y modificar el archivo hosts, pero para modificar el archivo necesitamos permisos de administrador por lo que primero abrimos el Bloc de Notas como administrador y abrimos la siguiente dirección C:\WINDOWS\system32\drivers\etc\hosts. Nos aparecerá de esta manera el archivo:

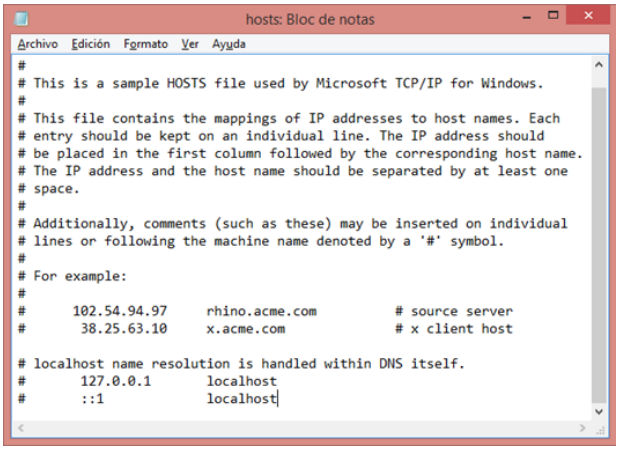

Figura 19: Configuración archivo hosts

En este archivo agregamos nuestro host virtual, para agregarlo lo hacemos de la siguiente manera:

IP Nombre de Host

Entonces nosotros agregaremos nuestros host apuntado a 127.0.0.1 que es la dirección IPv4 de la maquina local, y después el nombre de nuestro hosts. Podemos agregar los host que deseemos pero siempre apuntando a 127.0.0.1

127.0.0.1 pfc.dev

NOTA: Usamos pfc.dev pero podría ser cualquiera otro, lo único que esta dirección será la que habrá que meter en el navegador para que apunte a nuestro site local.

Paso 3:

Ahora debemos modificar el archivo de configuración de Apache, para incluir el archivo de configuración de virtual host, lo podemos abrir de igual manera con un bloc de notas.

En XAMPP, la ruta será la siguiente: C:\xampp\apache\conf\httpd.conf

Lo siguiente es buscar las siguientes dos líneas que están resaltadas:

Virtual host

Elimínese el # de la segunda línea.

Además dentro del mismo archivo debemos asegurarnos de que el módulo Rewrite está habilitado, para ello buscamos la siguiente línea:

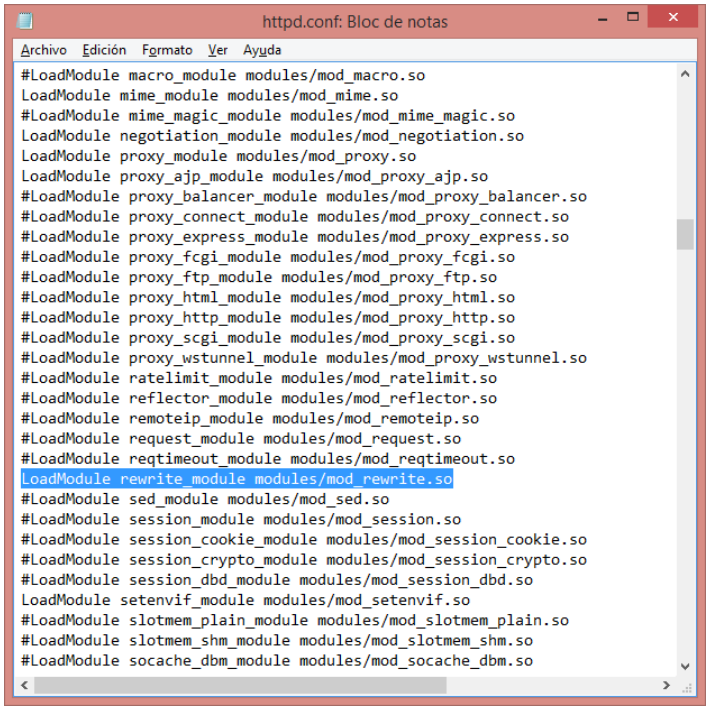

Figura 20: Configuración Rewrite en httpd.conf

httpd.conf

LoadModule rewrite_module modules/mod_rewrite.so

Y nos aseguramos de que no esté comentada (el signo de numeral # sirve para comentar líneas), si no tiene el signo quiere decir que ya está habilitada.

Realizado esto guardamos los cambios.

Paso 4:

Lo siguiente es abrir el archivo de configuración que nos provee XAMPP, de igual manera lo podemos editar con un Bloc de Notas.

Si están en XAMPP, la ruta será la siguiente: `C:\xampp\apache\conf\extra\httpd-vhosts.conf

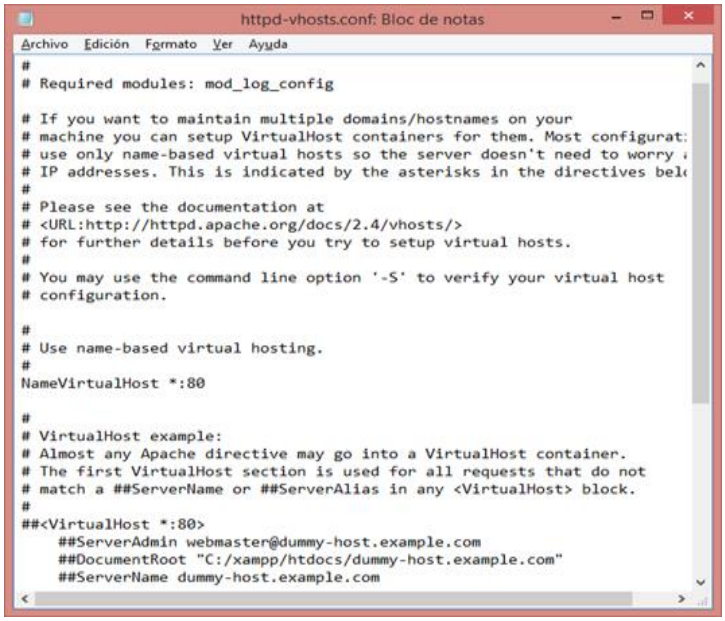

Figura 21: Configuración httpd-vhosts.conf

Es en este archivo donde alojaremos cada uno de los host virtuales que creemos, en esta caso nos valdrá con pegar este ya configurado para nuestro proyecto:

```
<VirtualHost *:80>
   ServerAdmin webmaster@pfc.com
   DocumentRoot c:\xampp\htdocs\pfc\dev
   ServerName pdf.dev
   <Directory "c:\xampp\htdocs\pfc\dev">
   Options Indexes FollowSymLinks
   Order allow,deny
   Allow from all
  </Directory>
</VirtualHost>
```

Es importante reemplazar c:\xampp\htdocs\pfc\dev y pfc.dev con su directorio y dominio local en caso de se que haya establecido uno diferente.

Nota: si se cambiado el puerto donde escucha Apache que por defecto es 80 a otro puerto (ejemplo: 8080), en ese caso ese el número de puerto que debes de poner en el encabezado de Virtual Host ejemplo: <VirtualHost *:8080>

Realizado todo esto procedemos a guardar nuestro archivo, y ahora nos toca reiniciar Apache, y probar el acceso al host virtual en el explorador en el caso de este ejemplo la dirección sería:

http://pfc.dev

8.1.5. Consola de Comandos

Abrimos en modo Ejecutar como Administrador la consola de git, GIT BASH que hemos instalado antes junto con Git. Esta consola tipo Unix, la usaremos para ejecutar los comandos necesarios para el desarrollo del proyecto así como para gestionar el control de versiones con Git.

Algunos comandos Básicos:

- Para saltar una carpeta hacia arriba al ser estilo Unix sería: cd ..
- Para saber en qué carpeta estamos: pwd
- Para listar los archivos en la carpeta dónde nos encontramos: ls
- Para crear una carpeta: mkdir
- Borrar rm
- Mover mv
- Copy paste cp
- Ir a Carpeta de usuario (home) cd ~/
- Ir a la raiz del ordenador cd /

8.1.6. Descargando el código desde Github

Para descargar el código nos dirigimos a Github que es el repositorio donde lo tenemos guardado y donde lo estamos versionando. Por motivos de desarrollo solo se guarda el código de desarrollo, los pases a producción se generarán al vuelo mediante comandos automatizados.

https://github.com/ilopezchamorro/pfc

Con la consola abierta lo primero es posicionarse en la carpeta que deseamos generar el código

cd c:

cd xampp/htdocs/pfc/dev

Una vez colocados en esta capeta hay que clonar el repositorio. Para esto ejecutamos el siguiente comando y se descargará de forma automática el código.

git clone https://github.com/ilopezchamorro/pfc.git .

NOTA: al final ponemos un punto, esto significa que nos descargue el código "aquí" (en esta carpeta) si no indicamos esto nos creará una carpeta padre y habría que cambiar el vhost que acabamos de configurar previamente.

Ya tenemos el código en nuestro ordenador.

8.1.7. Crear Base de Datos

Lo siguiente será crear la Base de Datos necesaria para el funcionamiento del proyecto.

Para crear la base de datos simplemente vamos a introducir en el navegador la siguiente ruta:

http://localhost/phpmyadmin

Esto nos dará acceso a PhpMyAdmin, un gestor de base de datos visual.

Si no se han dado credenciales específicas lo normal es que por defecto use:

User: root

Pass: (vacío, ninguna pass asignada)

Host: localhost

No es necesario cambiarlas para trabajar de forma local.

Una vez dentro tan solo se debe crear una nueva base de datos con el nombre que queramos.

Después pinchar en Importar, e importar el archivo sql.sql que está en la raiz del proyecto y la estructura de base de datos ya estará lista para trabajar, el sql que acabamos de importar ha generado nuestra base de datos.

8.1.8. ¿Por qué aun no puedo ver el proyecto en el navegador?

Si nos detenemos un momento podemos apreciar que hemos puesto una ruta que aún no existe para el virtual host.

c:/xampp/htdocs/pfc/dev

La carpeta dev la generará la aplicación al vuelo cuando ejecutemos los comandos para desarrollo.

8.1.9. Grunt

Necesitaremos instalar la línea de comandos de Grunt para que podamos ejecutarlos. En la consola de comandos (ejecutada en modo administrador):

npm install -g grunt-cli

Esto nos instala la línea de comandos de grunt de forma global en nuestro pc (tendremos acceso desde cualquier carpeta) gracias a -g

8.1.10. Npm

Node Package Modules. Es un repositorio de librerías de javascript.

Existe un archivo package.json en el que figuran todas las dependencias de desarrollo y de ejecución de javascript que nos hacen falta versionar ya que pueden

obtenerse desde el repositorio oficial ejecutando un solo comando, detectará qué librerías en qué versión queremos ya que lo tenemos configurado en dicho archivo.

Puede llevar un rato instalarlo todo hasta que descarga todas las librerías pero solo lo tendremos que realizar una vez, para esto ejecutar en la consola en modo administrador:

npm install

Automáticamente descargará todo lo necesario.

8.1.11. Bower

Bower es otro gestor de dependencias específico para front, en el que se encuentran librerías que no son sólo javascript sino también de css. Para este proyecto hemos utilizado un "reset" de estilo para igualar a todos los navegadores llamado Normalize. Como es CSS no está disponible en NPM así que usaremos Bower para instalarla.

Simplemente ejecutar el siguiente comando: bower install

8.1.12. Configuración de credenciales

En la raíz del proyecto existe un archivo secret.json.example

```
{
   "sftp":{
      "host" : "yourHost",
      "username" : "yourUser",
      "password" : "**********"
   },
   "mysql":{
      "dev":{
         "dbname": "dbname",
         "user" : "root",
         "pass" : "**********",
         "host" : "localhost"
      },
      "pro":{
         "dbname": "dbname",
         "user" : "exampleUser",
```

```
        "pass" : "**********",
        "host" : "localhost"
      }
   }
}
```

Sustituir los valores por los correctos y guardar el archivo con el nombre: secret.json. Este archivo nunca será versionado para no comprometer la seguridad.

- "sftp": credenciales para subir el código al servidor de producción de forma atomática. "mysql": credenciales de la base de datos en dos versiones:

- "dev": credenciales Base de Datos de desarrollo "pro": credenciales de la Base de Datos de producción

8.1.13. Comandos automatizados

Ahora sí estamos listos para interactuar con el proyecto. Si todo el proceso de instalación ha ido correctamente simplemente hay tres comandos básicos que se pueden usar para usar el proyecto:

- Desarrollo: *grunt*

Crea una caperta dev/, ejecuta browserify para inyección de dependencias de javascript, después compila y minifica javascript, después minifica y concatena CSS, mueve todos los archivos necesarios desde src hasta dev, abre la url en el navegador y se queda escuchando cambios en código para relanzar todo el proceso de forma automática al guardar un cambio y cambia las credenciales con las encontradas en el archivo secret.json de dev

- Generar código entregable: *grunt build*

Genera un entregable en un carpeta Build/ con todo el código listo para ejecutarse con las credenciales de pro.

- Pase a producción: *grunt deploy*

Hace lo mismo que el comando anterior pero además lo sube al servidor de producción los archivos.