ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA
INFORMÁTICA
INGENIERÍA DEL SOFTWARE

FIT & MEET – BACK-END Y APLICACIÓN WEB

FIT & MEET – BACK-END AND WEB APPLICATION

Realizado por
Nicolás Pozas García
Tutorizado por
Eduardo Guzmán De los Riscos
Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, junio de 2017

Fecha defensa:
El Secretario del Tribunal

Resumen: Se ha desarrollado una aplicación web, que permite, por una parte, a los usuarios que deseen realizar deporte la posibilidad de encontrar una instalación deportiva donde realizarlo, ofreciendo incluso la opción de permitir la reserva online del mismo, añadir los extras que ofrezcan las empresas, y toda la información GPS para poder llegar al destino correctamente.

Así mismo, podemos proporcionar a las empresas que ofrecen estos servicios la posibilidad de gestionar fácilmente sus instalaciones, permitiendo una gran flexibilidad en el horario y precio de estas. También permite recibir una retroalimentación de nuestros clientes, la confirmación de entradas mediante códigos QR, mostrar información sobre ellas, entre otras.

Por otra parte, se ha desarrollado una API siguiendo el protocolo OAuth2 que permite extender el uso de esta aplicación a dispositivos móviles y aplicaciones de terceros, ofreciendo las mismas posibilidades que a través de nuestra web, a nuestra aplicación móvil y a todas las aplicaciones de terceros que quieran utilizar nuestros servicios.

Palabras claves: instalaciones deportivas, gestión de reservas

Abstract: It has been developed a web application that allows, on one hand, to users who wish to do sports, the possibility of finding a sport facility where to do it, offering even the choice to allow the booking of it, adding extras that businesses offerors, and all GPS information to get the location of business successfully.

Likewise, we can give to businesses that offer these services the possibility of managing easily their facilities, allowing a great flexibility in the timetable and price of these, also allows to receive a feedback from our customers, the confirmation of reservations through QR codes, showing information about them, among others.

On the other hand, we have developed an API following the OAuth2 protocol that allows to extend the use of this application to mobile devices and third-party applications, offering the same possibilities that through our web, to our mobile application and all applications of third parties that want to use our services.

Keywords: sport facilities, booking management

CONTENIDO

1. INTRODUCCIÓN

1.1. Motivación

Este TFG cubre principalmente las necesidades de dos perfiles claramente diferenciados: por un lado tenemos el perfil del usuario que desea realizar alguna actividad deportiva y necesita reservar una pista para poder realizarla y, por otro lado, la gestión de los horarios, usuarios y pistas deportivas por parte de las empresas para ofrecer el servicio de reservas de una manera más eficiente, ofreciendo a los usuarios además de poder reservar la instalación desde la propia empresa, también de forma online.

1.2. Objetivos del TFG

El objetivo principal de este TFG es precisamente la construcción de una aplicación para solventar los problemas anteriores, tanto tener un directorio de todas las empresas con todos los deportes que ofrecen y poder consultar la disponibilidad de todas sus instalaciones deportivas, como la gestión de las empresas de sus instalaciones, pudiendo ofrecer diferentes horarios por pista, diferentes precios por hora y una serie opciones de lo más flexibles. Al estar este TFG desarrollado en grupo, podemos ofrecer más funcionalidad a este. Por un lado, vamos a tener una plataforma online donde poder realizar las siguientes acciones.

Para la plataforma online, tenemos 3 usuarios bien diferenciados, que son descritos a continuación:

- El usuario que desea realizar las actividades deportivas podrá:
    - Encontrar un lugar donde poder realizar deporte.
    - La posibilidad de poder reservarlo de manera online.
    - También tendrán un sistema de puntos acumulables con los que podrán recibir descuentos o algún día de reserva gratis.
- La empresa que desea ofrecer sus instalaciones deportivas podrá:
    - Administrar sus distintas instalaciones deportivas (reservas de pistas, estadísticas de reservas, etc.). Esto tiene una dificultad añadida, ya que al trabajar con muchas empresas diferentes y cada una tener unas necesidades concretas; por ejemplo, empresas que ofrezcan una pista donde se realicen dos deportes diferentes,

empresas que tienen alguna pista cubierta que no será cerrada en caso de que llueva, empresas que tengan algunas de sus instalaciones fuera de la propia empresa, o empresas que debido a algún motivo ofrezcan diferentes horarios y/o precios, para cada una de sus pistas, por comentar algunos ejemplos. Así pues, nuestra aplicación se adapta a cada una de estas situaciones.
    - Llevar un control de los usuarios que acceden a estas instalaciones.
- Por último, tenemos el perfil del administrador de la aplicación que será usado como soporte al perfil de la empresa, este perfil podrá:
    - Gestionar algunos cambios en las empresas, tales como los extras ofrecidos, nombres, descripciones y características básicas en general.
    - Añadir extras que las posibles empresas pueden ofrecer a sus usuarios.
    - Añadir deportes que las empresas pueden ofrecer a sus usuarios.
    - La gestión de los archivos multimedia que tienen las empresas y las instalaciones deportivas.

Como vemos este último perfil se encarga básicamente de orientar a la empresa y poder realizar pequeños cambios en ella, en caso de ser necesario.

Y, por otra parte, para la plataforma móvil, tenemos a un usuario; que será aquél que quiere realizar la reserva de una instalación deportiva. Este usuario podrá:

- Encontrar una instalación deportiva para realizar deporte, en este caso al tratarse de una plataforma móvil, tenemos mayor facilidad para ofrecer instalaciones por cercanía con la función GPS del dispositivo.
- La posibilidad de reservarlo desde cualquier parte.
- Y al igual que la plataforma web, contará con un sistema de puntos.

1.3. Tecnologías a usar

Como tenemos distintos subsistemas se va a explicar cada uno de forma independiente:

1.3.1. APLICACIÓN WEB

1.3.1.1. Lado del servidor

Para el desarrollo de esta aplicación web en el lado del servidor se han barajado varias opciones como pueden ser Python, Ruby, Java, PHP o NodeJS, entre muchas otras disponibles.

Se decidió usar PHP con respecto a Python o Ruby por la experiencia previa con esta plataforma, ya que Python o Ruby realmente Python o Ruby no ofrecen ninguna ventaja realmente interesante frente a PHP. Si bien es verdad que PHP "puro" no tiene una buena orientación a objetos y puede resultar un código ilegible al contrario que Python o Ruby, con el Framework usado Laravel, esta opción se soluciona.

Por otra parte, tenemos a Java, este lenguaje no se ha considerado ya que, al necesitar una máquina virtual para su funcionamiento, consume bastante más RAM que PHP en el servidor, haciéndolo más lento y pesado.

Realmente nuestra principal decisión para el lado del servidor ha sido entre PHP y NodeJS, si bien es verdad que NodeJS es quizás más rápido que PHP, y por defecto es asíncrono, no creemos que tenga la robustez que tiene PHP. Por otro lado, también es verdad que tengo más experiencia con el lenguaje PHP, y si hubiese realizado el proyecto en NodeJS no hubiese podido incluir tantos detalles y abarcar tanta funcionabilidad.

1.3.1.2. Lado del cliente

Para el lado del cliente hemos usado la tecnología HTML5, con JavaScript y CSS3, más concretamente como framework de JavaScript se ha usado JQuery, por ser el más usado a nivel mundial, y como framework de HTML5 y CSS3 Bootstrap, ya que ofrece una estructuración por columnas bastante buena, un diseño adaptativo que funciona bastante bien, y tiene una gran comunidad y documentación detrás de él.

1.3.2. API

Para la API hemos usado también PHP ya el Framework usado, Laravel, aunque no descartaría migrar a la tecnología NodeJS, por ofrecer las conexiones asíncronas, ser más rápido, y ofrecer más variedad de verbos HTTP, ofreciendo

una mayor legibilidad. PHP solo admite las acciones (GET y POST), mientras que NodeJS ofrece (GET, POST, PUT, DELETE, PATCH, entre otros).

Hemos usado el protocolo OAuth2, para ofrecer un método de conexión con las aplicaciones móviles. El uso de este protocolo se ha decidido por que ofrece una alta seguridad, y una gestión de tokens bastante eficiente.

1.3.3. BASES DE DATOS

En cuanto a la base de datos usada nos hemos decantado por MySQL, ya que se trata de una base de datos muy potente, muy usada, relacional y gratis. Podíamos haber elegido una base de datos como MongoDB, pero la aplicación es puramente relacional así que no nos parecía una buena opción. También podríamos haber elegido Oracle, pero no necesitamos tanta potencia.

1.3.4. SERVIDOR

Como servidor de la aplicación hemos usado ***Nginx*** ya que se trata de un servidor muy potente e ideal para webs realizas en PHP; otra opción hubiese sido por ejemplo ***Apache2.0*** que es otro fuerte líder del mercado. En este caso la elección ha sido por preferencia personal; antes usaba ***Apache2.0*** y esta vez hemos decido usar ***Nginx*** para aprender sobre otra tecnología.

1.4. Herramientas a usar

Para el desarrollo del código PHP se ha utilizado la herramienta ***PHPStorm***, pertenece a la compañía ***JetBrains*** y cuenta con muy buen soporte para PHP y la mayoría de sus Frameworks.

Para el servidor en local, se ha usado una máquina virtual llamada ***Homestead*** desarrollada por el propio creador de Laravel, esta máquina virtual viene con una serie de facilidades para la construcción de sitios web con Laravel, montados en el servidor ***Nginx***. Esta máquina virtual estará montada en ***VirtualBox***.

Para probar la API se ha utilizado la herramienta ***Postman*** que permite consultar URLS con todos los verbos posibles y crear una colección para la API.

Para los correos electrónicos se ha usado la web ***mailtrap.io***, esta web permite usar un servidor de correo electrónico donde se mandarán todos los correos electrónicos mandados por la aplicación.

1.5. Estructura de la memoria

La memoria se ha intentado estructura siguiendo el mismo orden en que se ha desarrollado el proyecto, si bien es cierto que se ha ido haciendo todas las partes en conjunto, más o menos se ha procedido en ese orden.

En el capítulo 2 hablaremos de cómo está estructurado el proyecto, que carpetas tenemos, que guardamos en cada carpeta, los casos de usos principales que pueden hacer los usuarios, y por último de la estructura de la base de datos.

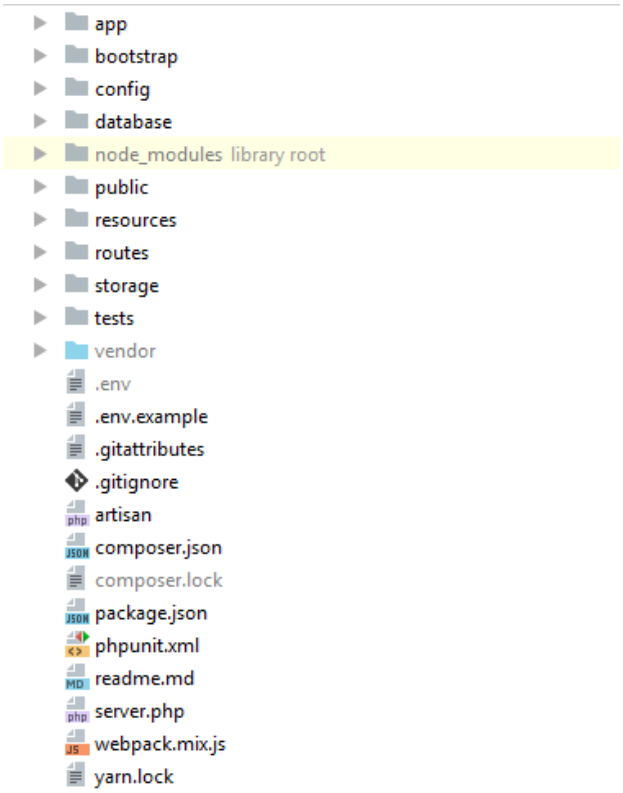
En el capítulo 3 se habla de la implementación en general del TFG, qué hemos usado, que problema nos ha surgido y como los hemos afrontado.

Por último, en el capítulo 4 hablamos sobre el resultado del TFG, que hemos aprendido, como se podría mejorar el proyecto, y qué hemos conseguido.

2. DISEÑO

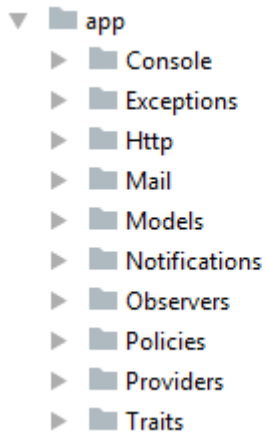
2.1. Estructura

Hemos usado el patrón de diseño de arquitecturas Modelo-Vista-Controlador, es una estructura simple que funciona bien en la mayoría de proyectos pequeños y medios, quizás en los grandes es mejor añadir más capas para evitar sobrecarga en los ficheros, en este caso, aunque nuestro proyecto no se considere grande, si es verdad que hemos tenido que dividir más la estructuración para que tampoco quede unas clases demasiado grandes, y para que se pueda escalar fácilmente. A continuación, vamos a mostrar los directorios de la aplicación y a hacer un breve resumen de cada uno.



2.1.1. APP

Este directorio contiene la lógica de la aplicación, vamos a explicar los directorios por encima, y nos meteremos más en detalle en los directorios que nos parezcan más importante.



El directorio ***Console*** contiene información sobre los comandos de consola de la aplicación, para nuestra aplicación no hemos usado estos comandos.

El directorio ***Exceptions*** contiene la información sobre las excepciones, Laravel viene configurado para procesar automáticamente las excepciones provenientes de códigos HTTP, esto quiere decir que si se crea un fichero con el código HTTP deseado se procesará y mostrará directamente esa vista. En nuestro caso hemos creado la vista ***resources/views/errors/404.blade.php***, así que, si en nuestra aplicación se produce un error 404, se mostrará directamente esta vista.

El directorio ***Http*** contiene varias carpetas para los controladores (***Controllers***), para los intermediarios de las rutas (***Middleware***) y para las peticiones (***Request***), estas carpetas son muy importantes así que las explicaremos cada una en su momento.

El directorio ***Mail*** contiene la lógica necesaria para el envío de los correos electrónicos de la aplicación, los correos electrónicos como tal se encuentran en la carpeta ***views***.

El directorio ***Models*** contiene todos los modelos usados por la aplicación, como también es una parte importante la comentaré más adelante.

El directorio ***Notifications*** contiene la lógica para el envío de todas las notificaciones, estas notificaciones no tienen por qué ser solamente mediante correo electrónico, es posible definir estas notificaciones por ***Slack***, ***Telegram*** y en general cualquier servicio.

El directorio ***Observers*** contiene diferentes observadores para los modelos, permitiendo realizar operaciones cuando los eventos sean escuchados.

El directorio ***Policies*** contiene la lógica que permite definir una mayor seguridad en la aplicación, cuando un usuario puede borrar un modelo, cuando actualizador, y en definitiva cualquier acción que se vaya a realizar en él.

El directorio ***Providers*** contiene varias clases que son usadas por Laravel, para definir ciertos comportamientos en la aplicación.

Y por último tenemos un directorio llamado ***Traits***, en este directorio guardamos ***Traits***, que para quien no lo sepa son una especie de comportamientos que pueden ser usadas por las diferentes clases de nuestro sistema para darle unas características comunes a una serie de clases.

2.1.2. BOOTSTRAP

Este directorio contiene toda la información de arranque de Laravel, simplemente contiene otros ficheros que se encargan de arrancar la aplicación.

2.1.3. CONFIG

Este directorio contiene archivos de configuración, tales como, la configuración general de la aplicación (***app.php***), la autenticación de los usuarios (***auth.php***), caché, bases de datos, y las diferentes librerías usadas en la aplicación.

2.1.4. DATABASE

Esta carpeta contiene tres carpetas más; la primera, ***factories***, es usada para definir modelos “falsos” para rellenar tablas, comúnmente llamado ***fakers***.

Contiene una segunda carpeta llamada ***migrations*** donde se almacenan todas las migraciones que hablaremos más adelante en el capítulo de bases de datos.

La última carpeta, llamada ***seeds,*** contiene las “semillas” de las bases de datos, que son usadas para introducir valores por defecto en la aplicación, se usan junto a los ***fakers*** para hacer los modelos de ejemplo más fácilmente y realistas.

2.1.5. NODE_MODULES

Esta carpeta contiene las dependencias de ***npm***, todas estas dependencias están definidas en el fichero ***package.json***.

2.1.6. PUBLIC

Como su nombre indica se almacena todos los ficheros públicos de la aplicación, como pueden ser las imágenes, los archivos de ***JavaScript***, los archivos ***CSS***, fuentes o el propio ***index.php***, que es donde debe apuntar el servidor.

2.1.7. RESOURCES

En este directorio se guardan principalmente 3 carpetas***, assets***, ***lang*** y ***views***.

En ***assets*** se guardan los archivos en crudo de la aplicación, como, por ejemplo:

- ***CSS*** sin minificar.
- ***JavaScripts*** en crudo.
- Y archivos ***SASS*** que se compilarán dando como resultado archivos ***CSS***.

En la carpeta ***lang*** se almacenan las traducciones de los textos estáticos de la aplicación, como los de los menús, mensajes, e incluso rutas.

En la última carpeta, ***views***, se almacenan todas las vistas de la aplicación, desde los paneles de administración, PDF, lo que se muestra al usuario, etc.

2.1.8. ROUTES

En esta carpeta se encuentra la configuración de las rutas de la aplicación, contiene los siguientes 4 ficheros:

- ***api.php*** → Define las rutas de la API.
- ***channels.php*** → Define las rutas de los canales de comunicación, como pueden ser chats.
- ***console.php*** → Define los comandos que pueden ser usados en la aplicación.
- ***web.php*** → Define todas las rutas dentro de la aplicación web.

Las rutas son un tema más complicado que abarcaremos más adelante.

2.1.9. STORAGE

En este directorio se guardan las vistas ya procesadas de la sintaxis ***blade*** a la sintaxis de PHP normal, se almacenan ***logs*** con lo que está ocurriendo en la aplicación, una carpeta “pública” donde almacenamos los perfiles de los usuarios, fotos de las empresas y todos los demás archivos multimedia que pueden subir los usuarios de la aplicación.

2.1.10. TESTS

Como su nombre indica en este directorio, se almacenan las pruebas unitarias, de integración y de navegación realizadas para nuestra aplicación, en nuestro caso no hemos realizado dichas pruebas.

2.1.11. VENDOR

En esta última carpeta se almacenan las librerías descargadas mediante ***composer***.

2.1.12. DIRECTORIO RAÍZ

En la raíz del directorio de carpetas se encuentra varios archivos importantes, entre ellos ***.env*** donde se configura algunos parámetros más delicados que no deben ser accesibles, como contraseñas.

El fichero ***artisan.php*** que contiene todos los comandos que pueden ser ejecutados en la aplicación.

El fichero ***composer.json*** que contiene la definición de los paquetes PHP de los cuales depende nuestra aplicación.

El fichero ***package.json*** como el anterior pero con las dependencias de ***npm***. Existen más ficheros, aunque estos son los más importantes.

2.2. Especificación y Análisis de requisitos

2.2.1. ANÁLISIS

Al tratarse de un TFG de elección propia, decidimos si realmente era necesario la realización de este sistema software; para ello se decidió realizar una encuesta con la ayuda de Google Forms, la encuesta consistía en 6 sencillas preguntas, para que fuese fácil de responder y fomentar a que se distribuyese lo máximo posible, ya que las encuestas más largas, suelen propagarse menos. Las preguntas fueron elegidas para obtener la máxima información, realizando las mínimas preguntas, estas preguntas fueron:

- ¿Qué edad tienes?
    - Menos de 18
    - 18 – 30
    - 30 – 40
    - 40 – 50
    - Más de 50

Con esta pregunta, podríamos enfocar la aplicación hacía un público más joven, con colores más llamativos y un lenguaje menos formal, o hacía un público más maduro, con unos colores más suaves y un lenguaje más formal.

- ¿Te gustaría reservar pistas deportivas a través del ordenador, móvil o Tablet?
    - Sí
    - No

Con esta pregunta, podríamos sacar si realmente sería relevante la aplicación en sí, ya que, si la mayoría de encuestados respondiesen que no, es que preferirían la opción de la reserva física, por tanto, no tendríamos cabida en este mercado.

- ¿Te descargarías una aplicación de móvil para reservarlas?
    - Sí
    - No

Con esta pregunta, vemos si es más necesario una aplicación nativa, o simplemente con una aplicación web, sería suficiente.

- ¿Qué sistema operativo usas en el móvil?
    - Android
    - iOS (Apple)
    - Otro…

Con esta pregunta, confirmamos cuál es la plataforma dominante, aunque ya sospechábamos que era Android, con esta encuesta queda confirmada, aunque iOS ha tenido un alto porcentaje, así que después del desarrollo en Android, sería interesante la opción del desarrollo en iOS.

- ¿Qué deporte prácticas?

En esta pregunta, pusimos los deportes más comunes, que se nos ocurrieron, aunque con el resultado de la encuesta nos dimos cuenta que Zumba, que no era una opción, tomada en cuenta, si que habría bastantes usuarios a los que les gustaría reservarlo por internet.

- ¿Cuántas horas de deporte realizas a la semana?
    - 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

Y, por último, con esta pregunta quisimos confirmar si realmente los usuarios a los que les gustaría este tipo de aplicación son los que más deporte realizan, ya

que normalmente los usuarios que realizan pocas horas a la semana (entre 0 y 4), nos dimos cuenta de que son los menos interesados en la aplicación, cosa que también sospechábamos.

2.2.1.1. Resultados

Como hemos dicho, la encuesta era muy rápida de contestar, y esa fue la clave de la gran propagación que tuvo, ya que, sólo la lanzamos a nuestros conocidos, y fue contestada por un total de 244 usuarios, dando un resultado de 30 deportes diferentes recopilados, y un uso de 3 plataformas móviles.

El resultado de la encuesta a las preguntas planteadas fue el siguiente:

- ¿Te gustaría reservar pistas deportivas a través del ordenador, móvil o Tablet?
    - Sí → 77.5%
    - No → 22.5%
- ¿Te descargarías una aplicación de móvil para reservarlas?
    - Sí → 71.3%
    - No → 28.7%
- Plataforma móvil más usada
    - Android → 79.7%
    - iOS → 19.9 %
- Deportes más comunes
    - Running → 22%
    - Musculación → 19.4%
    - Fútbol → 18.1 %
    - Pádel → 15.2%
    - Senderismo → 13.9%
    - Ciclismo → 11.3%
- Edad más propensa a realizar deporte y reservarlo
    - La edad más “atractiva” es de entre los 18 a los 40 años. Por tanto, la aplicación debería ir orientada a un público joven.

Se adjuntarán unas transparencias con un análisis más detallado.

2.2.2. REQUISÍTOS

2.2.2.1. Funcionales

Aquí se van a listar los requisitos funcionales, que son los que determinan la funcionabilidad de la aplicación y definen las acciones que pueden realizar los diferentes roles de la misma. Las listas están divididas según los diferentes roles de usuarios:

Rol	*ID*	*Nombre*	*Descripción*
Usuario base (de este extienden los demás)	UN1	Poder entrar al sistema	El usuario debe poder entrar en el sistema.
	UN2	Poder salir del sistema	El usuario debe poder salir del sistema.
	UN3	Modificar la contraseña	El usuario debe poder modificar su contraseña.
	UN5	Buscar deporte.	El usuario podrá buscar una empresa para poder realizar el deporte especificado.
	UN5	Reservar deporte	El usuario podrá reservar una instalación deportiva donde realizar el deporte.
	UN6	Contactar con la empresa	El usuario podrá contactar con la empresa para sugerir mejoras, o solucionar problemas.
	UN7	Registro del sistema.	El usuario podrá registrarse en el sistema para poder realizar las reservas y manejar su perfil.

Rol	*ID*	*Nombre*	*Descripción*
	AA1	Administrar usuarios	El usuario podrá realizar un CRUD completo de los usuarios.
	AA2	Administrar roles	El usuario podrá realizar un CRUD completo de los roles.
	AA3	Administrar permisos	El usuario podrá realizar un CRUD completo de los permisos

Administrador de la aplicación	AA4	Administrar deportes	El usuario podrá realizar un CRUD completo de los deportes que se pueden practicar en la aplicación
	AA5	Administrar países	El usuario podrá realizar un CRUD completo de los países donde ofertamos deportes.
	AA6	Administrar municipios	El usuario podrá realizar un CRUD completo de los municipios donde ofertamos deportes.
	AA7	Administrar empresas	El usuario podrá realizar un CRUD completo de las empresas que ofertan deportes.
	AA8	Administrar extras de empresas	El usuario podrá realizar un CRUD completo de los extras de empresas.
	AA9	Administrar instalaciones	El usuario podrá realizar un CRUD completo de las instalaciones deportivas donde se practican los deportes.
	AA10	Ver registro de actividad	El usuario podrá ver el registro de actividad completo de la aplicación.

Rol	*ID*	*Nombre*	*Descripción*
Administrador de empresa	AE1	Administrar empresa	El usuario puede cambiar cualquier aspecto de su propia empresa.
	AE2	Administrar instalaciones	El usuario podrá realizar un CRUD completo de todas sus instalaciones.
	AE3	Administrar deportes	El usuario podrá realizar un CRUD completo de los deportes que ofrece en sus instalaciones.

AE4	Realizar reservas	El usuario podrá realizar reservas en caso de que el usuario base, quiera hacerlo de manera física.
AE5	Confirmar reservas	El usuario podrá confirmar las reservas realizadas por los usuarios bases.
AE6	Seguimiento de reservas	El usuario podrá llevar un seguimiento de las reservas realizadas en su empresa.
AE7	Administrar lógicas de horario	El usuario podrá administrar las lógicas de horarios de sus instalaciones en realización con un deporte.
AE8	Administrar lógicas de precio	El usuario podrá administrar las lógicas de precios de sus instalaciones en realización con un deporte.

2.2.2.2. No funcionales

Categoría	ID	Descripción
Seguridad	RNF1	Todos los datos entre las peticiones deberán ir validados.
Seguridad	RNF2	No se permitirá el acceso al panel de administración a los usuarios que no sean administradores de la aplicación (error 404).
Seguridad	RNF3	No se permitirá el acceso al panel de administración de empresas a los usuarios que no sean administradores de empresa (error 404).
Seguridad	RNF4	Para el uso de la API será necesario siempre pasar el token de acceso mediante la cabecera **Authorization**.

Usabilidad	RNF5	El sistema informará al usuario de cualquier error producido durante sus acciones, mediante alertas que aparecerán en pantalla.
Usabilidad	RNF6	Se les facilitará a los usuarios un calendario para poder realizar las reservas más cómodamente.
Accesibilidad	RNF7	Los usuarios podrán acceder desde Chrome, Firefox e Internet Explorer.

2.3. Casos de uso

2.3.1. DESCRIPCIÓN DE LOS CASOS DE USO

Vamos a pasar a describir los escenarios más importantes de la aplicación, aunque no son todos.

ID	*U1*
Caso de uso	Realizar la entrada al sistema
Actores	Usuario no autenticado
Descripción	Acceso típico a la aplicación
Escenario principal	1. El usuario intenta acceder a cualquier página con acceso restringido de la aplicación. 2. El usuario es redirigido a la página de entrada al sistema. 3. El usuario introduce sus datos correctamente en el sistema. 4. El usuario es redirigido a la página a la que estaba intentando acceder.
Escenario alternativo	3. Alguno de los datos introducidos por el usuario no es correcto, o el usuario no tiene cuenta. 4. El usuario vuelve a la página de entrada al sistema, donde los campos erróneos están resaltados.

ID	*U2*
Caso de uso	Realizar la salida del sistema
Actores	Usuario autenticado
Descripción	Cierre de sesión del usuario
Escenario principal	1. El usuario pulsa sobre el botón de salir. 2. El usuario cierra sesión y es redirigido a la página principal.

ID	*U3*
Caso de uso	Modificar mi contraseña
Actores	Usuario autenticado
Descripción	Actualizar mi contraseña
Escenario principal	1. El usuario pulsa en el botón de perfil del menú superior. 2. El usuario es redirigido a su perfil. 3. El usuario pulsa en el botón de cambiar contraseña. 4. El usuario es redirigido al formulario de cambiar contraseña. 5. El usuario introduce su contraseña nueva y la confirmación. 6. El usuario ha cambiado su contraseña, se le muestra un mensaje y se le redirige a la página del perfil
Escenario alternativo	6. Se produce un error al intentar cambiar la contraseña, se le redirige al formulario de cambio de contraseña y se le muestra el error producido.

ID	*U4*
Caso de uso	Buscar una empresa para realizar deporte
Actores	Cualquier usuario
Descripción	El usuario puede buscar una empresa donde poder hacer deporte.
Escenario principal	1. El usuario puede seleccionar una localidad y/o deporte. 2. El sistema busca las empresas que coinciden con esta búsqueda y las muestra en un listado.
Escenario alternativo	2. Se muestra un listado vacío si no se ha encontrado ninguna empresa que coincida con la búsqueda.

ID	*U5*
Caso de uso	Reservar un deporte
Actores	Usuario autenticado
Descripción	El usuario puede reservar una instalación deportiva para realizar un deporte.
Escenario principal	1. El usuario puede buscar una empresa. 2. Se muestra un listado con las empresas que coincidan con el criterio indicado. 3. El usuario puede pulsar en el botón detalles. 4. El usuario es redirigido a la información sobre la empresa. 5. El usuario puede elegir el deporte a realizar. 6. El sistema muestra las pistas donde se pueden realizar ese deporte. 7. El usuario puede pulsar en disponibilidad de la pista. 8. El usuario es redirigido al comienzo de la reserva.

	9. El usuario puede seleccionar una franja horaria donde realizar la reserva y darle al botón de ir a por los extras. 10. El usuario es redirigido al listado de los extras. 11. El usuario puede seleccionar los extras relacionados con esa pista y darle a ver resumen de la reserva. 12. El usuario es redirigido al resumen de su reserva. 13. El usuario revisa la reserva y pulsa en confirmar. 14. El usuario recibe un correo electrónico y es redirigido al inicio de la aplicación.
Escenario alternativo	2. No se encuentra ninguna empresa que coincida con el criterio de búsqueda. 3. El usuario debe introducir un nuevo criterio. 6. El usuario no encuentra ninguna pista donde poder realizar deporte. 7. El usuario debe buscar otra empresa. 9. No se ha encontrado ninguna franja horaria para poder hacer deporte. 10. El usuario debe buscar otra pista. 11. No se ha encontrado ningún extra, para poder reservarlo. 12. El usuario debe buscar otra pista que tenga ese extra. 14. La reserva no se pudo confirmar. 15. Se anula la reserva. 14. El correo electrónico no se ha podido mandar. 15. El usuario tendrá que meterse en el perfil a descargar su reserva.

ID	*U6*
Caso de uso	Contactar con la empresa
Actores	Cualquier usuario
Descripción	El usuario quiere contactar con la empresa.
Escenario principal	1. El usuario pulsa en el menú superior en contacto. 2. El usuario es redirigido a la página de contacto de la empresa. 3. El usuario rellena el formulario de contacto. 4. El sistema manda el correo electrónico de contacto y el usuario es redirigido a la página de contacto, donde se le muestra el mensaje.
Escenario alternativo	4. No se pudo enviar el correo electrónico. 5. Se redirige al usuario a la página de contacto, donde se le muestra al usuario un mensaje sobre el error producido.

ID	*U7*
Caso de uso	Un usuario puede registrarse en el sistema.
Actores	Cualquier usuario
Descripción	El usuario puede crearse una cuenta para utilizar la aplicación
Escenario principal	1. El usuario pulsa en el botón superior entrar. 2. El usuario es redirigido al formulario de entrada en el sistema. 3. El usuario pulsa en el enlace “¿Todavía no tienes cuenta? Haz deporte con nosotros.” 4. El usuario es redirigido al formulario de registro 5. El usuario introduce sus datos en el formulario. 6. El sistema crea una cuenta para el usuario, y manda un correo electrónico dándole la bienvenida. 7. El usuario está dentro del sistema.

Escenario alternativo	6. El usuario ha introducido un correo electrónico ya registrado, o su contraseña es demasiado corta. 7. El usuario será redirigido al formulario con los mensajes con los errores producidos. 6. No se puede enviar el correo de bienvenida al usuario.

Los demás casos de uso que se pueden producir en la aplicación proceden de los usuarios super administradores o administradores de empresas. Estos casos de uso corresponden a CRUD de los diferentes elementos de la aplicación, así que no se consideran tan importante como para especificarlos, siendo similares a los ya indicados.

2.3. Base de datos

Como hemos comentado, hemos elegido ***MySQL*** para la base de datos, para gestionarla hemos usado el programa ***HeidiSQL***, ya que es totalmente gratis, y ofrece una correcta administración para ver el estado de las tablas, añadir datos y ver el almacenamiento que ocupa cada tabla.

Nuestro sistema cuenta con un total de 32 tablas, la definición de estas tablas se ha desarrollado mediante los esquemas de migraciones en Laravel, vamos a explicar uno más o menos complejo de forma rápida para que quede claro cómo funcionan.

2.3.1. MIGRACIONES

Las migraciones son clases que extienden de la clase ***Migration***:

```php
<?php

use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateReservationsTable extends Migration {

    /**
     * Table name
     * @var string
     */
    protected $tableName = 'reservations';
```

Para todas estas clases hemos definido una propiedad con el nombre de la tabla, ya que se reutilizará en la clase.

Estas clases tienen dos métodos principales, el método ***up*** que se ejecutará cuando se ejecute la migración y contiene la propia definición de la tabla, como vemos a continuación

```php
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up() {

        if(!Schema::hasTable($this->tableName)) {

            Schema::create($this->tableName, function(Blueprint $table) {

                $table->bigIncrements( column: 'id');
                $table->dateTime( column: 'start');
                $table->dateTime( column: 'end');
                $table->enum( column: 'status_code', [
                    'pending',
                    'confirm',
                    'refused'
                ])->default('pending');

                $table->timestamps();
            });

            Schema::table($this->tableName, function(Blueprint $table) {

                $table->integer( column: 'user_id')->unsigned()->nullable()->default(null);
                $table->foreign( columns: 'user_id')->references('id')->on('users')->onDelete('cascade')->onUpdate('cascade');

                $table->integer( column: 'facility_sport_id')->unsigned()->nullable()->default(null);
                $table->foreign( columns: 'facility_sport_id')->references('id')->on('facilities_plays_sports')->onDelete('cascade')->onUpdate('cascade');

                $table->index([
                    'id',
                    'user_id'
                ]);

                $table->unique([
                    'start',
                    'facility_sport_id'
                ]);

                $table->unique([
                    'end',
                    'facility_sport_id'
                ]);
            });
        }
    }
```

Existen varias definiciones, en este caso, tenemos los siguientes:

- ***bigIncrements:*** que corresponde a un número entero grande que se irá auto incrementando.

- ***dateTime:*** que corresponde a una fecha completa con el siguiente formato AAAA-MM-DD HH:MM:SS.
- ***enum:*** que corresponde a un enumerado y como segundo parámetro le pasaremos los posibles valores.
- ***timestamps***: que corresponde a dos campos ***dateTime*** uno llamado ***created_at*** y otro llamado ***updated_at*** donde se almacenará cuando se crear y cuando se actualiza la fila.

Como vemos también podemos definir claves foráneas con el método ***foreign,*** índices con el método ***index*** y campos únicos con el método ***unique***.

El otro método principal es ***down*** que se ejecutará cuando se revierta la migración, el caso más habitual como este es borrar la tabla en caso de que exista.

2.3.2. TABLAS DEL SISTEMA

Como hemos comentado tenemos un total de 32 tablas, las cuales vamos a describir brevemente a continuación.

2.3.2.1. activity_log

En esta tabla se almacena un registro de todos los cambios que se están realizando en los objetos del sistema, para tener toda la información lo más controlada posible. Sus columnas son:

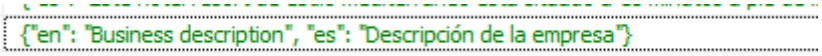
- ***ID*** → Identificador del registro.
- ***LOG_NAME*** → Nombre del registro.
- ***DESCRIPTION*** → Qué acción se ha desarrollado (creación, actualización, borrado, etc.).
- ***SUBJECT_ID*** → Cuál es el identificador del objeto del registro.
- ***SUBJECT_TYPE*** → Clase que corresponde al objeto del registro.
- ***CAUSER_ID*** → Identificador del causante del registro (quién ha realizado la acción).
- ***CAUSER_TYPE*** → Clase del causante del registro.
- ***PROPERTIES*** → Corresponde a un JSON con las propiedades anteriores y las propiedades nuevas después del cambio.

- ***CREATED_AT*** → Cuando se creó el registro. Como hemos comentado antes esta columna podría decirse que es común a todas las tablas, así que solo lo comentaremos en esta, y se extenderá para las demás.
- ***UPDATED_AT*** → Cuando se actualizó el registro. Pasará lo mismo que con la columna anterior.

2.3.2.2. businesses

En esta tabla se almacena todas las empresas que están registradas en nuestra aplicación, y que ofrecerán sus servicios. Sus columnas son:

- ***ID*** → Identificador de la empresa.
- ***NAME*** → Nombre de la empresa.
- ***DESCRIPTION*** → Una descripción de la empresa. Esta columna al igual que muchas otras que veremos más adelante tiene una particularidad, y es que en lugar de ser un campo de texto se trata de un campo JSON, donde habrá pares de claves/valor; esto se ha especificado así para permitir que la aplicación sea multi-idioma. De esta forma puede extender a todos los idiomas que queramos sin modificar ninguna parte importante de la aplicación; la gestión de esto la explicaremos más adelante, mientras tanto podemos ver la siguiente imagen que contiene un pequeño ejemplo:



- ***SLUG*** → Este campo es una modificación del campo ***NAME*** ya que lo que se hace es ponerlo todo en minúsculas, separado por guiones en lugar de espacios, y quitando los caracteres especiales. Este campo es usado para mejorar el posicionamiento SEO de la web, ya que en lugar de poner un número (el ***ID*** de la empresa) podemos poner su nombre.
- ***ADDRESS*** → Dirección física de la empresa.
- ***LATITUDE*** → Latitud de la empresa. Usado junto con la longitud para ofrecer varias funciones de Geolocalización, ofertas por cercanías, y demás.
- ***LONGITUDE*** → Longitud de la empresa.
- ***PHONE*** → Teléfono de la empresa
- ***EMAIL*** → Dirección de correo electrónico para contactar con la empresa.

- ***MUNICIPALITY_ID*** → A qué municipio pertenece la empresa. Usado para las búsquedas.
- ***OWNER_ID*** → Identificador del usuario que puede administrar la empresa desde nuestro sistema.

2.3.2.3. businesses_extras

En esta table almacenamos toda la información sobre los posibles extras que pueden ofrecer las empresas. Sus columnas son:

- ***ID*** → Identificador del extra.
- ***NAME*** → Nombre del extra con la posibilidad de multi idioma (JSON).
- ***ICON*** → Icono para dar una muestra visual del extra, para que quede más estético.

2.3.2.4. businesses_have_extras

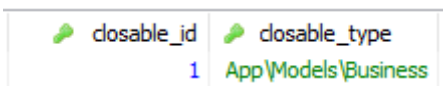
Esta tabla simplemente es la relación intermedia entre las empresas y sus extras, ya que varias empresas pueden ofrecer los mismos extras. Sus columnas son:

- ***ID*** → Identificador de la relación (en este caso no es necesario, pero por costumbre lo suelo poner).
- ***BUSINESS_ID*** → Identificador de la empresa.
- ***BUSINESS_EXTRA_ID*** → Identificador del extra de empresa.

2.3.2.5. closed_days

En esta tabla se guardará el día en que cierra una empresa o una instalación, ya que puede ser que la empresa se mantenga abierta, pero la instalación esté cerrada por reformas, por ejemplo. Sus columnas son:

- ***ID*** → Identificador del día de cierre.
- ***DATE*** → Fecha de cierre
- ***CLOSABLE_ID*** → Identificador del objeto a cerrar.
- ***CLOSABLE_TYPE*** → Tipo de objeto a cerrar. Estas dos columnas se unen para formar lo que se llama una relación polimórfica, es decir, tenemos el identificador del objeto y el tipo de objeto y podemos sacar toda la información del objeto. Un ejemplo de esto podemos verlo a continuación:



Esa fila hace referencia a la empresa con identificador 1, este tipo de relaciones son muy útiles ya que permiten extender la funcionabilidad de la aplicación de una manera muy fácil y sin ningún cambio en la base de datos.

2.3.2.6. comments

En esta tabla se almacenan los comentarios que dejan los usuarios sobre las empresas y las instalaciones deportivas, si bien los comentarios de las instalaciones deportivas ahora mismo no se muestran, pueden hacerlo en un futuro. Sus columnas son:

- ***ID*** → Identificador del comentario.
- ***RATING*** → Valoración del usuario.
- ***COMMENT*** → Comentario del usuario.
- ***PUBLISHED*** → Estado del comentario. Esta columna permite despublicar un comentario en caso de que sea ofensivo, sea spam o no deba estar publicado por cualquier otro motivo.
- ***COMMENTABLE_ID*** → Identificador del objeto a comentar.
- ***COMMENTABLE_TYPE*** → Tipo del objeto a comentar.
- ***USER_ID*** → Usuario que ha realizado el comentario.

2.3.2.7. countries

En esta tabla se almacena la información sobre los países de la aplicación, ahora mismo no se usa, pero está pensado por si la aplicación se amplia, podemos tener la posibilidad de filtrar por países. Sus columnas son:

- ***ID*** → Identificador del país.
- ***NAME*** → Nombre del país
- ***LATITUDE*** → Latitud del país. Pensamos en la capital del país para poder centrarlo en un mapa.
- ***LONGITUDE*** → Longitud del país.

2.3.2.8. facilities

En esta tabla se almacenan todas las instalaciones deportivas que las empresas ofrecen para que los usuarios puedan reservarlas. Sus columnas son:

- ***ID*** → Identificador de la instalación
- ***NAME*** → Nombre de la instalación. Preparado para el multi idioma (JSON).
- ***LATITUDE*** → Latitud de la instalación para poder geolocalizarla.
- ***LONGITUDE*** → Longitud de la instalación para poder geolocalizarla.
- ***BUSINESS_ID*** → Empresa a la que pertenece esta instalación.

2.3.2.9. facilities_plays_sports

Esta tabla es quizás la tabla más importante del sistema, ya que relaciona cada instalación con los deportes que se pueden realizar en ella, así como un precio base y la duración del deporte. Sus columnas son:

- ***ID*** → Identificador de la relación.
- ***MINUTES*** → Minutos que tarda el deporte en realizarse.
- ***PRICE*** → Precio base del deporte. Este precio podrá ser incrementado con una tabla que veremos más adelante.
- ***PRICE_PER_PERSON*** → Esta columna es un booleano que permite saber si el deporte se paga por persona o es un precio global.
- ***SPORT_ID*** → Identificador del deporte que se realiza.
- ***FACILITY_ID*** → Identificador de la instalación deportiva donde se realiza.

2.3.2.10. facility_sport_extras

Tabla donde se almacenan los extras que puede tener un deporte en una instalación, se ha hecho de esta manera para darle más flexibilidad a la empresa, ya que puede darse el caso de que en una instalación un deporte tenga un extra y en otro no. Sus columnas son:

- ***ID*** → Identificador de la relación
- ***NAME*** → Nombre del extra, multi idioma (JSON).
- ***DESCRIPTION*** → Descripción del extra, multi idioma (JSON).
- ***PRICE*** → Cuánto vale el extra.
- ***FACILITY_SPORT_ID*** → Identificador del deporte en la instalación. Como se verá más adelante la tabla de facilities_plays_sports tiene mucha importancia y en la aplicación se toma como una clase propia.

2.3.2.11. media

Tabla donde se almacena toda la información de las fotos de las instalaciones, de las empresas y en general se puede extender a cualquier entidad de la aplicación. Sus columnas son:

- ***ID*** → Identificador del archivo multimedia.
- ***MODEL_ID*** → Identificador del modelo.
- ***MODEL_TYPE*** → Tipo del modelo.
- ***COLLECTION_NAME*** → Nombre de la colección donde se guardará ese archivo multimedia.
- ***NAME*** → Nombre del archivo multimedia. Es el nombre que se muestra.
- ***FILE_NAME*** → Nombre del fichero multimedia. Este si es el nombre del fichero almacenado.
- ***MIME_TYPE*** → Tipo del fichero multimedia.
- ***DISK*** → En qué disco está almacenado. Esto suena raro, pero en Laravel se pueden crear “discos virtuales” y este campo sirve para saber en qué disco virtual está guardado el fichero.
- ***SIZE*** → Cuánto ocupa el fichero multimedia.
- ***MANIPULATIONS*** → Algunas manipulaciones que se hayan hecho al archivo.
- ***CUSTOM_PROPERTIES*** → Propiedades definidas por nosotros para dar más información al archivo.
- ***ORDER_COLUMN*** → Es un orden para mostrar los ficheros.

2.3.2.12. migrations

Esta es una tabla principal de Laravel, donde se almacenan las migraciones que tiene nuestra aplicación, y sirve para administrarlas. Sus columnas son:

- ***ID*** → Identificador de la migración.
- ***MIGRATION*** → Nombre de la migración.
- ***BATCH*** → Esta columna es usada para llevar un control de las migraciones hechas y que quedan por hacer. Por ejemplo, al principio todas las tablas migradas al principio tienen un 1, si migramos una tabla nueva se le asignará un 2, esto significa que al deshacer la migración se deshará la más alta.

2.3.2.13. municipalities

En esta tabla se almacenará los municipios de la aplicación, así se facilita las búsquedas por municipios, funciones de geolocalización y demás. Sus columnas son:

- ***ID*** → Identificador del municipio.
- ***NAME*** → Nombre del municipio.
- ***LATITUDE*** → Latitud del municipio (del centro del municipio).
- ***LONGITUDE*** → Longitud del municipio.
- ***COUNTRY_ID*** → Identificador del país.

2.3.2.14. oauth_access_tokens

En esta tabla se almacenará los tokens de acceso de todos los clientes. Sus columnas son:

- ***ID*** → Identificador del token de acceso.
- ***USER_ID*** → A qué usuario se le ha asignado este token.
- ***CLIENT_ID*** → A qué cliente corresponde este token.
- ***NAME*** → Nombre del token de acceso.
- ***SCOPES*** → El ámbito del token de acceso. Este ámbito se usa para dar más o menos permisos a los diferentes clientes.
- ***REVOKED*** → Si el token de acceso sigue estando vigente.
- ***EXPIRES_AT*** → Cuando expira el token de acceso.

2.3.2.15. oauth_auth_codes

En esta tabla se almacena los códigos de autenticación para los clientes que usen redirección OAuth, en nuestro caso se mantendrá vacía. Sus columnas son:

- ***ID*** → Identificador del token de acceso.
- ***USER_ID*** → A qué usuario se le ha asignado este código.
- ***CLIENT_ID*** → A qué cliente corresponde este código.
- ***SCOPES*** → El ámbito del código de autorización. Este ámbito se usa para dar más o menos permisos a los diferentes clientes.
- ***REVOKED*** → Si el código de autorización sigue estando vigente.
- ***EXPIRES_AT*** → Cuándo expira el código de autorización.

2.3.2.16. oauth_clients

En esta tabla se almacena todos los clientes OAuth que tendrá nuestra aplicación. Sus columnas son:

- ***ID*** → Identificador del cliente.
- ***USER_ID*** → Usuario al que pertenece este cliente.
- ***NAME*** → Nombre del cliente.
- ***SECRET*** → Secreto del cliente.
- ***REDIRECT*** → Al tratarse de un servicio OAuth algunos clientes pueden querer redirigir a su página después de la autenticación.
- ***PERSONAL_ACCESS_CLIENT*** → Es un campo booleano que indica si el cliente es con una cuenta personal, esto solo se utilizará para las empresas, en nuestra aplicación no se usará.
- ***PASSWORD_CLIENT*** → Si es un cliente de tipo contraseña, estos clientes son usados para aplicaciones móviles y para aplicaciones de terceros, este será el que más usemos en nuestra aplicación, empezando por uno para Android, uno para iOS, y algunos más para aplicaciones de terceros que quieren trabajar con nosotros.
- ***REVOKED*** → Campo booleano que indica si el cliente sigue vigente.

2.3.2.17. oauth_personal_access_clients

En esta tabla se almacena los clientes de acceso personal que se han creado en nuestro sistema. Sus columnas son:

- ***ID*** → Identificador del cliente de acceso personal.
- ***CLIENT_ID*** → A qué cliente corresponde este acceso personal.

2.3.2.18. oauth_refresh_tokens

En esta tabla se almacena la información sobre los tokens de refresco. Sus columnas son:

- ***ID*** → Identificador del token de refresco.
- ***ACCESS_TOKEN_ID*** → Token de acceso.
- ***REVOKED*** → Campo booleano que indica si el token ha sido rechazado o no.
- ***EXPIRES_AT*** → Cuándo expirará el token de refresco.

2.3.2.19. password_resets

Esta tabla contiene información sobre los reseteos de las contraseñas, cuando un usuario indica que se le ha olvidado la contraseña y se le manda el correo para reiniciarla. Sus columnas son:

- ***EMAIL*** → Correo electrónico que ha solicitado el reinicio de la contraseña.
- ***TOKEN*** → Es un token que contiene la información sobre cuando expira la posibilidad de reinicio de la contraseña. Por defecto es 60 minutos.

2.3.2.20. permissions

Tabla que contendrá los permisos que podrán tener los usuarios. Sus columnas son:

- ***NAME*** → Nombre del permiso.

2.3.2.21. price_variations

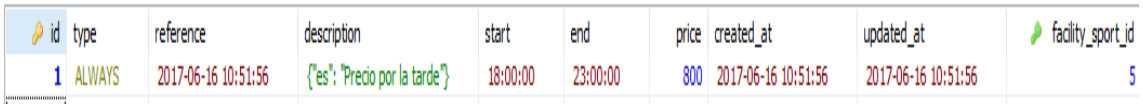
En esta tabla se almacena la lógica de la variación de precios. Es una tabla en principio un poco complicada de entender, así que vamos a ver primero las columnas y después explicaré la funcionalidad:

- ***TYPE*** → Tipo de lógica, este tipo de lógica es un enumerado que admite todas las combinaciones de fechas posibles que se nos ha ocurrido. Los valores pueden ser:
  - ***ALWAYS, DAY, DAY_AND_MONTH, DAY_OF_WEEK, DAY_OF_MONTH, WEEK, WEEK_OF_MONTH, WEEK, WEEK_OF_MONTH, MONTH, MONTH_AND_YEAR*** y ***YEAR.***
- ***REFERENCE*** → Fecha de referencia que se tomará en cuanto para realizar la lógica.
- ***DESCRIPTION*** → Descripción de la lógica. Multi idioma (JSON)
- ***START*** → Hora a la que empieza a aplicarse la lógica.
- ***END*** → Hora a la que termina de aplicarse la lógica.
- ***PRICE*** → El incremento o decremento del precio que se aplica.
- ***FACILITY_SPORT_ID*** → Identificador del deporte en la instalación.

Una vez descritas las columnas, podemos ver un pequeño ejemplo del funcionamiento de las lógicas. Imaginemos que tenemos en nuestra base de

datos que se puede practicar fútbol en la pista 1, por 12€ como precio base todo el día.

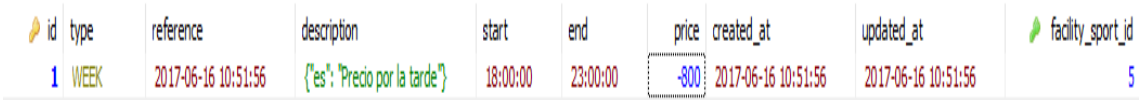
Ahora vamos a introducir la siguiente lógica:



id	type	reference	description	start	end	price	created_at	updated_at	facility_sport_id
1	ALWAYS	2017-06-16 10:51:56	{"es": "Precio por la tarde"}	18:00:00	23:00:00	800	2017-06-16 10:51:56	2017-06-16 10:51:56	5

Con esa lógica podemos indicarle que a partir de las 6 de la tarde hasta las 11 de la noche el precio se incrementará en 800 céntimos (8€), costando así 20€.

Veamos otro ejemplo para tenerlo más claro:



id	type	reference	description	start	end	price	created_at	updated_at	facility_sport_id
1	WEEK	2017-06-16 10:51:56	{"es": "Precio por la tarde"}	18:00:00	23:00:00	-800	2017-06-16 10:51:56	2017-06-16 10:51:56	5

Tenemos esta lógica además de la anterior, dando un resultado que desde el día 2017-06-12 hasta el día 2017-06-18 (la semana que corresponde al día 2017-06-16, que es el que se toma como referencia), por la tarde costará 8€ menos, anulando la lógica anterior solo para esta semana quedando todo el día el precio a 12€. Sabemos que es algo difícil de entender a simple vista, pero podemos ver que es una herramienta muy potente para ajustar perfectamente los precios con una precisión increíble para las empresas y totalmente flexible.

2.3.2.22. reservations

Tabla donde se almacenan las reservas realizas por la aplicación. Sus columnas son:

- ***ID*** → Identificador de la reserva.
- ***START*** → Día y hora a la que se empezará a jugar.
- ***END*** → Día y hora a la que se terminará de jugar.
- ***STATUS_CODE*** → Enumerado con el estado en que se encuentra la reserva. Si la reserva se ha realizado por el usuario, hasta que el administrador de la empresa no escanee el código QR no se confirmará; si la reserva se hace directamente por el administrador de la empresa, estará confirmada directamente.
- ***USER_ID*** → Quién ha realizado la reserva.
- ***FACILITY_SPORT_ID*** → En qué deporte e instalación se jugará.

2.3.2.23. reservation_items

En esta tabla se almacenará todos los elementos que se incluyan en la reserva, tanto la reserva principal como los extras elegidos. Sus columnas son:

- ***ID*** → Identificador del elemento.
- ***SKU*** → Texto identificador del elemento.
- ***PRICE*** → Precio por unidad de este elemento.
- ***TAX*** → Impuestos que se le aplican.
- ***CURRENCY*** → En qué moneda se especifica, ahora mismo solo se especifica en euros.
- ***QUANTITY*** → Cantidad de elementos a reservar.
- ***ITEM_ID*** → Identificador del elemento.
- ***ITEM_TYPE*** → Tipo del elemento.
- ***RESERVATION_ID*** → A qué reserva pertenece.

2.3.2.24. roles

En esta tabla se almacena los roles que pueden tener los usuarios. Sus columnas son:

- ***ID*** → Identificador del rol.
- ***NAME*** → Nombre del rol.

2.3.2.25. role_has_permissions

En esta tabla se almacena la relación entre los permisos que tienen los roles. Sus columnas son:

- ***PERMISSION_ID*** → Identificador del permiso.
- ***ROLE_ID*** → Identificador del rol.

2.3.2.26. schedule_variations

Esta tabla al igual que la que contenía la lógica del precio, contiene la lógica del tiempo, funciona de la misma forma, pero con una prioridad, dando lugar así a que, si una lógica tiene una prioridad más alta que otra, la de menor prioridad queda anulada. Sus columnas son:

- ***ID*** → Identificador de la lógica.
- ***TYPE*** → Tipo de lógica.
- ***DESCRIPTION*** → Descripción de la lógica. Multi idioma (JSON)

- ***START*** → Hora de inicio en que se empieza a aplicar la lógica.
- ***END*** → Hora de final en la que la lógica pierde el efecto.
- ***PRIORITY*** → Prioridad de la lógica.
- ***FACILITY_SPORT_ID*** → Identificador del deporte en la instalación.

2.3.2.27. sports

En esta tabla se almacenan todos los posibles deportes que se pueden realizar a través de la web. Sus columnas son:

- ***ID*** → Identificador del deporte.
- ***NAME*** → Nombre del deporte. Multi idioma (JSON).
- ***ICON*** → Icono representativo del deporte.

2.3.2.28. users

Tabla donde se almacenan todos los usuarios del sistema. Sus columnas son:

- ***ID*** → Identificador del usuario.
- ***NAME*** → Nombre del usuario.
- ***EMAIL*** → Correo electrónico del usuario. Este campo ha de ser único.
- ***PASSWORD*** → Contraseña encriptada del usuario.
- ***AVATAR*** → Avatar que mostrar el usuario.
- ***DESCRIPTION*** → Una descripción personal breve.
- ***GENDER*** → Sexo para recibir en un futuro información personalizada.
- ***POINTS*** → Puntos que lleva acumulados el usuario. Cada céntimo de reserva equivale a un punto, y con estos puntos se pueden aplicar descuentos.
- ***REMEMBER_TOKEN*** → Este es el token que veíamos anteriormente en la tabla de ***password_resets.***

2.3.2.29. users_like_municipalities

En esta tabla se almacenan las localidades donde el usuario tiene preferencia por realizar el deporte. Sus columnas son:

- ***ID*** → Identificador de la relación.
- ***USER_ID*** → Identificador del usuario.
- ***MUNICIPALITY_ID*** → Identificador del municipio.

2.3.2.30. users_like_sports

En esta tabla se almacenan los deportes preferidos por el usuario. Sus columnas son:

- ***ID*** → Identificador de la relación
- ***USER_ID*** → Identificador del usuario.
- ***SPORT_ID*** → Identificador del deporte.

2.3.2.31. user_has_permissions

En esta tabla se almacenan los permisos que tiene el usuario. Sus columnas son:

- ***USER_ID*** → Identificador del usuario.
- ***PERMISSION_ID*** → Identificador del permiso.

2.3.2.32. user_has_roles

En esta tabla se almacenan los roles que tiene el usuario. Sus columnas son:

- ***USER_ID*** → Identificador del usuario.
- ***ROLE_ID*** → Identificador del rol.

3. IMPLEMENTACIÓN

En este capítulo se hablará tanto sobre del desarrollo de la aplicación como de los problemas surgidos, y sus soluciones. Ahora vamos a pasar a comentar los puntos más importantes del desarrollo.

3.1. Controladores y rutas

Los controladores juegan uno de los papeles más importantes en Laravel y en cualquier framework bajo el patrón de diseño modelo-vista-controlador (MVC). Los controladores por lo general cumplen con el MVC, y tienen métodos para listar el modelo ***(index)***, ver el formulario de creación de un modelo ***(create)***, almacenar un modelo ***(store)***, mostrar información sobre un modelo ***(show)***, mostrar el formulario para editar el modelo ***(edit)***, actualizar un modelo ***(update)*****,** eliminar un modelo ***(destroy)***. Como vemos estos métodos suelen tener una sintaxis clara que corresponde con una arquitectura ***REST***. Por eso mismo a la hora de definir las rutas podemos definirlas como:

- ***Route::get*** → Que corresponde con una petición mediante el verbo ***GET.***
- ***Route::post*** → Que corresponde con una petición mediante el verbo ***POST***.
- ***Route::put*** → Que corresponde con una petición mediante el verbo ***PUT***.
- ***Route::patch*** → Que corresponde con una petición mediante el verbo ***PATCH.***
- ***Route::delete*** → Que corresponde con una petición mediante el verbo ***DELETE.***
- ***Route::options*** → Que corresponde con una petición mediante el verbo ***OPTIONS.***

Laravel al igual que todos los frameworks de PHP tiene un problema con las acciones que no sean ***GET*** y ***POST***, y es que no existen en PHP. Por tanto, para imitarlas lo que se hace es utilizar realmente el verbo ***POST*** pero añadir un campo oculto llamado _***method*** donde se indicará el verbo por el que tiene que pasar, un ejemplo sería ***_method = "PUT"***. Esto para el funcionamiento normal de la aplicación no supone un problema, pero para la API si lo ha hecho. Ya que, para la API, lo que hemos tenido que hacer es usar todos los verbos ***PUT,***

PATCH y ***DELETE*** como verbos ***POST***, esto no supone un gran cambio, pero la arquitectura ***REST*** queda menos legible.

Las rutas, se definen de la siguiente forma:

```
Route::put('comments/{comment}/change-publish', 'CommentController@putChangePublish')->name('comments.change-publish');
```

Como vemos, se especifica primero la acción, después un ***string*** que corresponde con la ruta a mostrar en la barra de direcciones***.*** En este caso, vemos que tenemos ***{comment}***, esta sintaxis indica que en la URL aparecerá un parámetro llamado ***comment***; como segundo parámetro se pasará el método del controlador encargado de realizar la opción oportuna. También usamos el método ***name***, este método acepta un ***string*** que será el identificador de la ruta que será el que usemos para llamarla desde los diferentes sitios.

Otra opción interesante que permiten estas rutas es la de agruparlas para tener unas características similares, por ejemplo, podemos ver el siguiente grupo:

```
Route::group(['prefix' => 'dashboard', 'middleware' => ['auth', 'admin'], 'namespace' => 'Dashboard'], function() {
```

A estas agrupaciones podemos añadirles un prefijo de ruta, definir que cumplan una serie de ***middleware*** que se explicarán más adelante y un ***namespace*** a partir del cual se formarán los nombres de los controladores indicados. Por ejemplo, en este caso tenemos la anterior ruta de los comentarios dentro de esta agrupación de rutas, por tanto, tenemos que:

La URL final será ***/dashboard/comments/{comment}/change-publish***.

El nombre del controlador será ***Dasboard\CommentController***.

Y para poder acceder a ellas debe cumplir que el usuario esté autenticado, y que el usuario sea administrador.

Un ejemplo de estas clases controlador es:

```php
/**
 * Store a newly created resource in storage.
 *
 * @param StoreUser $request
 * @return \Illuminate\Http\Response
 */
public function store(StoreUser $request) {

    $user = new User();
    $user->setAttribute('name', $request->input( key: 'name'));
    $user->setAttribute('email', $request->input( key: 'email'));
    $user->setAttribute('password', bcrypt($request->input( key: 'password')));
    $user->setAttribute('description', $request->input( key: 'description'));
    $user->setAttribute('gender', $request->input( key: 'gender'));

    if($request->hasFile( key: 'avatar')) {

        $avatar = $request->file( key: 'avatar')->store( path: 'users',  options: 'public');
        $user->setAttribute('avatar', $avatar);
    }

    $saved = $user->save();

    $password = $request->input( key: 'password');

    if($saved) {

        // Relations
        $user->roles()->sync($request->input( key: 'roles'));
        $user->sports()->sync($request->input( key: 'sports'));
        $user->municipalities()->sync($request->input( key: 'municipalities'));

        \Mail::to($user->getAttribute( key: 'email'))->send(new UserCreatedMail($user, $password));

        $request->session()->flash('success', __('Usuario creado correctamente'));

    } else {

        $request->session()->flash('danger', __('No se pudo crear el usuario'));
    }

    return redirect()->route( route: 'users.index');
}
```

Este método de un controlador es usado para almacenar un usuario en el sistema. Al método se le pasa un parámetro llamado ***$request*** del tipo ***StoreUser***, estas peticiones las comentaremos también más adelante, en el apartado de seguridad.

Hay poco más que mencionar: se crea un nuevo usuario, se guarda en la base de datos, si se puede guardar, se crea un mensaje en sesión con el título ***Usuario creado correctamente.*** Como vemos este título está también dentro de un método ***__()***, este método sirve para traducir los textos de la aplicación y comentaremos en el apartado de traducciones.

También mandamos un mensaje al usuario, esos mensajes se indicarán más adelante en el apartado comunicaciones.

Por último, devolvemos una redirección a la ruta llamada ***users.index***. En otros métodos, lo que hacemos es devolver una vista con la función:

```
return view( view: 'dashboard.users.create', compact( varname: 'roles', _: 'sports', 'municipalities'));
```

Esta función tiene dos parámetros: el primero es el nombre de la vista, y el segundo es un array con todos los datos que queremos pasar a la vista para tenerlos disponibles.

3.2. Entidades

En este apartado, vamos a hablar de las entidades de la aplicación, estas entidades comprenden los modelos, los observadores, y los ***traits***.

3.2.1. MODELOS

El sistema cuenta con un total de 15 modelos. Como hemos comentado en más de una ocasión vamos a comentar un modelo complejo y vamos a suponer que los demás modelos son similares.

3.2.1.1. Atributos

```php
class User extends Authenticatable {

    use Notifiable, HasRoles, LogsActivity, HasApiTokens;

    /**
     * Table
     *
     * @var string
     */
    protected $table = 'users';

    /**
     * The attributes that are mass assignable.
     *
     * @var array
     */
    protected $fillable = [
        'name', 'email', 'password', 'avatar', 'description', 'gender', 'points'
    ];

    /**
     * The attributes that are logged.
     * @var array
     */
    protected static $logAttributes = [
        'name', 'email', 'password', 'avatar', 'description', 'gender', 'points'
    ];

    /**
     * The attributes that should be hidden for arrays.
     *
     * @var array
     */
    protected $hidden = [
        'password', 'remember_token'
    ];

    /**
     * The attributes that should be casting
     * @var array
     */
    protected $casts = [
        'name' => 'string',
        'email' => 'string',
        'avatar' => 'string',
        'description' => 'string',
        'gender' => 'string',
        'points' => 'integer',
    ];
```

Al principio de los modelos definimos los ***traits***, y una serie de propiedades de clases:

- ***$table*** → Tabla que representa el modelo.

- ***$fillable*** → Atributos que vamos a permitir almacenarse de forma masiva.
- ***$logAttributes*** → Atributos en los que nos interesa controlar los cambios, estos atributos serán registrados por la tabla de registro cuando se modifiquen.
- ***$hidden*** → Estos atributos estarán ocultos cuando se envíe el modelo mediante una petición JSON.
- ***$casts*** → Este array es usado para convertir los atributos a los datos dados.

3.2.1.2. RELATIONSHIPS

Estas funciones representan relaciones entre los modelos:

```php
public function sports() {

    return $this->belongsToMany( related: Sport::class, table: 'users_like_sports', foreignKey: 'user_id', relatedKey: 'sport_id')->withTimestamps();
}

public function municipalities() {

    return $this->belongsToMany( related: Municipality::class, table: 'users_like_municipalities', foreignKey: 'user_id', relatedKey: 'municipality_id')->withTimestamps();
}

public function comments() {

    return $this->hasMany( related: Comment::class, foreignKey: 'user_id', localKey: 'id');
}

public function reservations() {

    return $this->hasMany( related: Reservation::class, foreignKey: 'user_id', localKey: 'id');
}

public function business() {

    return $this->hasOne( related: Business::class, foreignKey: 'owner_id', localKey: 'id');
}

public function facilities() {

    return $this->hasManyThrough( related: Facility::class, through: Business::class, firstKey: 'owner_id', secondKey: 'business_id', localKey: 'id');
}
```

Tenemos varios tipos de relaciones: las relaciones muchos a muchos ***(belongsToMany)***, uno a muchos ***(hasMany)***, uno a uno ***(hasOne)*** y relaciones muchos a muchos a través de ***(hasManyThrough)***. Estos métodos tienen una serie de parámetros, por ejemplo, con qué clase está relacionada, qué tabla está involucrada, qué columnas de las tablas intervienen, etc. Estos métodos devuelven una colección de objeto de Laravel.

3.2.1.3. SCOPES

Este tipo de funciones son extensiones de la funcionabilidad de ***Eloquent*** de Laravel, este se encarga de la conexión de los modelos con la base de datos. Un ejemplo de ***scopes*** es:

```php
public function scopePublished($query) {

    return $query->where('published', true);
}
```

Al poner este método en el modelo ***Comment*** que representa a un comentario en la aplicación, podemos usar lo siguiente:

Comment::published()->get()

Que indica que quiero todos los comentarios que se encuentran publicados.

3.2.1.4. MUTATORS

Por último, tenemos los ***mutators*** que son una variación de los atributos normales, este concepto se explica mejor con un ejemplo:

```php
public function getLatitudeAttribute($value): float {

    if(is_null($value)) {

        $value = $this->business->getAttribute('latitude');
    }

    return $value;
}
```

Como vemos es una funciona sencilla. Esta función pertenece al modelo ***Facility***, que representa a una instalación deportiva; al definir este método podemos usar lo siguiente:

$facility->latitude

Ese atributo será convertido, para ello primero se invocará a la función anterior, y se aplicará lo que pone, que en caso de que la instalación no tenga definida la latitud, se usará la latitud de la empresa, a la que pertenezca.

3.2.2. OBSERVADORES

Los observadores son una parte importante de los modelos, ya que permiten activar funciones que escuchan eventos provocados en el sistema, por ejemplo, en el observador de la clase de usuario tenemos lo siguiente:

```php
class UserObserver {

    /**
     * Listen to the User created event.
     *
     * @param  User  $user
     * @return void
     */
    public function created(User $user) {

        $user->notify(new WelcomeUserNotify());
    }

    /**
     * Listen to the User deleting event.
     *
     * @param User $user
     * @return void
     */
    public function deleting(User $user) {

        if(Storage::disk('public')->exists($user->getAttribute( key: 'avatar'))) {

            Storage::disk('public')->delete($user->getAttribute( key: 'avatar'));
        }
    }
}
```

Este observado tiene dos métodos, el primero se activará cuando se haya creado al usuario ***(created)***, el segundo método se activará justo en el momento anterior a eliminar el usuario ***(deleting)***.

Cuando el usuario ha sido creado se le enviará una notificación de bienvenida. Y cuando el usuario vaya a ser eliminado se procederá a eliminar su avatar del servidor.

Una vez definidos estos observadores es necesario registrarlos en el fichero ***App\Providers\AppServiceProvider***:

```php
class AppServiceProvider extends ServiceProvider {

    /**
     * Bootstrap any application services.
     *
     * @return void
     */
    public function boot() {

        Business::observe( class: BusinessObserver::class);
        BusinessesExtra::observe( class: BusinessExtraObserver::class);
        Facility::observe( class: FacilityObserver::class);
        Reservation::observe( class: ReservationObserver::class);
        Sport::observe( class: SportObserver::class);
        User::observe( class: UserObserver::class);
    }
```

3.2.3. TRAITS

Los ***traits*** son una especie de clases que ofrecen una funcionabilidad común. En nuestro sistema hemos creado dos ***traits*** que son: ***Geolocalized*** y ***Translateable***.

El ***trait Geolocalized***, tiene una función, una que calcula la distancia en kilómetros entre dos modelos que sean geolocalizables.

El otro ***trait***, ***Translateable***, tiene una función que permite traducir los atributos de una clase, a continuación, vamos a verlo y comentarlo.

```php
trait Translateable {

    public function translateAttribute(string $attribute): string {

        // Lang
        $lang = app()->getLocale();
        $fallbackLocale = config( key: 'app.fallback_locale');

        // Localized
        $localized = '';
        $original = $this->getAttribute($attribute);

        if(array_key_exists($lang, $original)) {

            $localized = $original[$lang];

        } else if(array_key_exists($fallbackLocale, $original)) {

            $localized = $original[$fallbackLocale];
        }

        return $localized;
    }
}
```

```php
public function getLocalizedNameAttribute() : string {

    return $this->translateAttribute( attribute: 'name');
}
```

Como ya hemos visto anteriormente algunas clases están preparadas para la traducción de algunos de sus métodos; en este caso vamos a comentar el ***mutator*** que obtiene el nombre del modelo ***Sport***. Le vamos a pasar al método ***translateAttribute*** el ***string 'name'*** que corresponde con el nombre del deporte. El método ***translateAttribute*** recibirá esa cadena de texto y buscará primero si existe el atributo en el idioma actual de la aplicación; de no ser así, buscamos si ese atributo se encuentra en el idioma por defecto de la aplicación y por último en caso de que no se encuentre en ninguno de los dos casos anteriores devolveremos la cadena de texto vacía.

3.3. Traducciones

Ya hemos comentado el ***trait*** de traducción, lo que queda por comentar de las traducciones es el sistema que con este fin usa Laravel. Para ello todas las cadenas de texto que queramos traducir las pasaremos por la función ***__('cadena-de-texto')***. Esta función buscará en el archivo que corresponde con el idioma actual de la aplicación; en caso de no encontrarse devolverá el idioma por defecto de la aplicación y, por último, en caso de no encontrase, se devolverá la cadena de texto sin procesar.

Veámoslo con un ejemplo: la aplicación se encuentra en inglés ***(en)***, el idioma por defecto de la aplicación es el español ***(es)***, y hemos llamado al método ***__('Crear')***. Esta llamada primero intentará buscar en el fichero ***/resources/lang/en.json***, la traducción de ***'Crear'***. Si no se encuentra, lo buscará en el fichero ***/resources/lang/es.json***, y, en caso de no encontrarse tampoco en este fichero, se devolverá la cadena de texto ***'Crear'***.

3.4. Comunicaciones

Laravel posee una potente herramienta para comunicarse con un usuario, para ello primero tenemos que definir una notificación. Todas estas notificaciones, pueden ser encoladas, para enviarse cuando el sistema esté más libre de carga.

3.4.1. NOTIFICACIONES

```php
class WelcomeUserNotify extends Notification {

    use Queueable;

    /**
     * Create a new notification instance.
     *
     */
    public function __construct() {

    }

    /**
     * Get the notification's delivery channels.
     *
     * @param  mixed  $notifiable
     * @return array
     */
    public function via($notifiable) {

        return ['mail'];
    }

    /**
     * Get the mail representation of the notification.
     *
     * @param  mixed  $notifiable
     * @return WelcomeUserMail
     */
    public function toMail($notifiable) {

        return (new WelcomeUserMail($notifiable))->to($notifiable->email, $notifiable->name);
    }

    /**
     * Get the array representation of the notification.
     *
     * @param  mixed  $notifiable
     * @return array
     */
    public function toArray($notifiable) {

        return [
            //
        ];
    }
}
```

Esta notificación representa cuando un usuario es registrado en el sistema y queremos darle la bienvenida, concretamente en este caso solo se procesará una notificación mediante correo electrónico. Aunque podrían definirse múltiples sistemas de notificaciones: ***trello, slack, telegram, sms, twitter, facebook***, y muchísimas más. Para ello tan solo haría falta modificar el array que se devuelve en el método ***via***, y añadir un método con la lógica para poder enviar la notificación. Por ejemplo, si queremos enviar una notificación por ***telegram***, tendríamos que devolver por el método vía un array con los dos valores: ***mail*** y

telegram; y añadir un método ***toTelegram*** con la lógica necesaria para poder enviar la notificación.

3.4.2. MAILABLES

Un ejemplo de ***mailable*** es el correo de bienvenida a un usuario:

```php
class WelcomeUserMail extends Mailable {

    use Queueable, SerializesModels;

    public $user;

    /**
     * Create a new message instance.
     *
     * @param User $user
     */
    public function __construct(User $user) {

        $this->user = $user;
    }

    /**
     * Build the message.
     *
     * @return $this
     */
    public function build() {

        return $this->markdown( view: 'emails.users.welcome')->subject(__('Bienvenido a Fit & Meet'));
    }
}
```

Como vemos, tenemos un constructor donde guardaremos todos los parámetros que queramos tener disponibles en la vista como públicos.

Y un método que sirve para construir el propio correo electrónico, como vemos le hemos pasado una vista e indicamos que va a ser construida mediante lenguaje ***markdown***, el asunto de este mensaje será ***'Bienvenido a Fit & Meet'***.

3.5. Seguridad

3.5.1. MIDDLEWARE

Esta herramienta es sumamente importante, ya que son pequeñas clases que actúan de intermediarias para las rutas, son especialmente útiles para ofrecer una mayor seguridad al sistema, Laravel tiene varios ***middleware*** predefinidos y en uso, pero vamos a hablar principalmente de los que hemos creado nosotros.

Todos estos ***middleware*** deben estar definidos en el fichero ***App\Http\Kernel.php***, un ejemplo de este fichero es:

```php
    /**
     * The application's route middleware.
     *
     * These middleware may be assigned to groups or used individually.
     *
     * @var array
     */
    protected $routeMiddleware = [
        'admin' => IsAdmin::class,
        'ajax' => AjaxRequest::class,
        'auth' => Authenticate::class,
        'auth.basic' => AuthenticateWithBasicAuth::class,
        'bindings' => SubstituteBindings::class,
        'business-admin' => IsBusinessAdmin::class,
        'can' => Authorize::class,
        'guest' => RedirectIfAuthenticated::class,
        'throttle' => ThrottleRequests::class,
        'localize' => LaravelLocalizationRoutes::class,
        'localizationRedirect' => LaravelLocalizationRedirectFilter::class,
        'localeSessionRedirect' => LocaleSessionRedirect::class
    ];
```

3.5.1.1. AjaxResquest

```php
class AjaxRequest {

    /**
     * Handle an incoming request.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  \Closure  $next
     * @return mixed
     */
    public function handle($request, Closure $next) {

        if(!$request->ajax()) {

            abort( code: 404);
        }

        return $next($request);
    }
}
```

Este ***middleware*** obliga a que las rutas que estén protegidas con él, deban accederse obligatoriamente mediante una petición ***ajax.***

3.5.1.2. IsAdmin

```php
class IsAdmin {

    /**
     * Handle an incoming request.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  \Closure  $next
     * @return mixed
     */
    public function handle($request, Closure $next) {

        if(auth()->check() && auth()->user()->hasRole('administrator')) {

            return $next($request);
        }

        abort( code: 404);
    }
}
```

Este ***middleware*** obliga a que la ruta protegida con él, puedan acceder solo los usuarios que esté autenticados y que tenga el rol de ***administrator***.

3.5.1.3. IsBusinessAdmin

```php
class IsBusinessAdmin {

    /**
     * Handle an incoming request.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  \Closure  $next
     * @return mixed
     */
    public function handle($request, Closure $next) {

        if(auth()->check() && auth()->user()->hasRole('business-admin') && !is_null(auth()->user()->business)) {

            return $next($request);
        }

        abort( code: 404);
    }
}
```

Este ***middleware*** obliga a que la ruta protegida con él, solo puedan acceder los usuarios que estén autenticados y tengan el rol de ***business-admin***.

3.5.2. REQUEST

Las ***request*** son el mecanismo por el cual se puede recoger información dentro de un controlador. Además, en nuestro caso hemos definido unas ***request*** personalizadas que ofrecen una validación adicional**.** En nuestro sistema tenemos unas 40 de estas validaciones, así que no vamos a proceder a explicar

todas, ya que son muy similares; vamos a explicar una que es más completa y las demás serán parecidas.

La ***request*** a explicar será ***UpdateUser***, esta ***request*** se usará cuando se desee actualizar a un usuario, las ***request*** están divididas en dos partes: ***authorize*** y ***rules***.

3.5.2.1. AUTHORIZE

```php
/**
 * Determine if the user is authorized to make this request.
 *
 * @return bool
 */
public function authorize() {

    return auth()->user()->can('update', $this->route( param: 'user'));
}
```

El método ***authorize*** tiene que devolver un valor booleano, si devuelve ***true*** el usuario actual estará autorizado, de lo contrario, se devolverá un error 403 de ***Action Unauthorized***.

Para este caso revisamos las políticas de la clase, con el usuario que aparece en el parámetro ***user***. Estas políticas ofrecen una seguridad añadida que explicaremos un poco más adelante.

3.5.2.2. RULES

Este método ofrece una serie de reglas que deben cumplirse si queremos que el proceso continúe, en caso de que alguna de las reglas no se cumpla simplemente se redirigirá al usuario a la anterior vista indicando los campos que no cumplen con las reglas indicadas.

```php
/**
 * Get the validation rules that apply to the request.
 *
 * @return array
 */
public function rules() {

    return [
        'name' => [
            'required',
            'min:3',
            'max:255'
        ],
        'email' => [
            'required',
            'email',
            'max:255',
            Rule::unique( table: 'users',  column: 'email')
        ],
        'password' => [
            'nullable',
            'min:6'
        ],
        'avatar' => [
            'nullable',
            'image',
            Rule::dimensions()->minWidth( value: 300)->minHeight( value: 300)
        ],
        'description' => [
            'nullable',
            'min:3',
            'max:255',
            'string'
        ],
        'gender' => [
            'required',
            Rule::in(['male', 'female'])
        ],
        'roles' => [...],
        'roles.*' => [...],
        'sports' => [...],
        'sports.*' => [...],
        'municipalities' => [
            'array'
        ],
        'municipalities.*' => [
            Rule::exists( table: 'municipalities',  column: 'id')
        ],
    ];
}
```

En estas reglas, comprobamos longitudes mínimas de campos: si un campo es requerido o no, las dimensiones que debe contener una imagen, que un campo tenga que estar en una serie de valores, comprobar que un campo sea único en

una base de datos, o comprobar que exista previamente. Existen muchas más reglas, pero ya están documentadas en la documentación oficial de Laravel.

3.5.3. POLÍTICAS

Las políticas son definidas una por modelo, en total contamos con 16 políticas de seguridad. Estas clases de políticas contienen varios métodos, uno por cada situación que queramos controlar. Por ejemplo, si queremos ver cuándo un usuario puede publicar un comentario, podemos usar el método ***create***; para seguir con el anterior ejemplo veíamos, cuando un usuario podía actualizar a otro usuario, y para ello se llamaba al método ***update.***

```
/**
 * Determine whether the user can update the user.
 *
 * @param User $me
 * @param \App\Models\User $user
 * @return mixed
 */
public function update(User $me, User $user) {

    return $me->getAttribute( key: 'id') === $user->getAttribute( key: 'id');
}
```

Como vemos, este método devolverá ***true*** si el usuario al que está intentando modificar es él mismo. Todas estas políticas tienen un método especial llamado ***before***.

```
/**
 * Check first
 * @param $user
 * @param $ability
 * @return bool
 */
public function before($user, $ability) {

    if($user->hasRole('administrator')) {

        return true;
    }
}
```

Este es un método especial, ya que es el primero que se comprueba al buscar una política. Si se satisface la condición que establece este método, todas las acciones relacionadas serán cumplidas; en caso contrario,, no se podrá realizar

ninguna acción. Si no se devuelve nada por este método, ya es cuando se comprueba si se puede realizar la acción concreta.

Como vemos si quien está intentando actualizar al usuario es el administrador del sistema, podrá hacerlo sin problemas, de lo contrario, solo podrá actualizarse si es él mismo quien está intentando actualizarse.

Todas estas políticas deben estar registradas en el fichero ***App\Providers\AuthServiceProvider***.

```
    /**
     * The policy mappings for the application.
     *
     * @var array
     */
    protected $policies = [
        Activity::class => ActivityPolicy::class,
        Business::class => BusinessPolicy::class,
        BusinessesExtra::class => BusinessesExtraPolicy::class,
        Comment::class => CommentPolicy::class,
        Country::class => CountryPolicy::class,
        Facility::class => FacilityPolicy::class,
        FacilitySport::class => FacilitySportPolicy::class,
        FacilitySportExtra::class => FacilitySportExtraPolicy::class,
        Municipality::class => MunicipalityPolicy::class,
        Permission::class => PermissionPolicy::class,
        PriceVariation::class => PriceVariationPolicy::class,
        Reservation::class => ReservationPolicy::class,
        Role::class => RolePolicy::class,
        ScheduleVariation::class => ScheduleVariationPolicy::class,
        Sport::class => SportPolicy::class,
        User::class => UserPolicy::class
    ];
```

3.6. API

Como ya hemos comentado anteriormente, la API está creada usando el protocolo ***OAuth2***, para poder utilizar dicho protocolo tendremos que crear mínimo un cliente que use la identificación por contraseña y no por redirección. En nuestro caso hemos creado dos clientes, uno para ***iOS*** y otro para ***Android***; si quisiéramos diferenciar otro cliente sería tan fácil como crearlo mediante el comando:

php artisan passport:client –password

Al ejecutar este comando, preguntará un nombre para identificar este cliente. Una vez introducido el nombre devolverá un identificador y un secreto, con estos

datos podemos comunicarnos con la API. Las rutas de comunicación están indicadas en la colección de ***POSTMAN***, que se adjuntará con el código. Tan solo habrá que sustituir los datos proporcionados, por los que se hayan generado con el anterior comando.

4. CONCLUSIONES

Este capítulo está dedicado a las conclusiones obtenidas durante el desarrollo del TFG.

4.1. Resultado final

Durante el desarrollo de este TFG hemos tenido una serie de problemas y es que, al ser una idea definida por nosotros, ha sufrido muchos cambios y hemos transformado el TFG sin querer durante su desarrollo. Al principio estaba planteado de una forma más social y se ha ido convirtiendo en una herramienta más destinada a la administración para empresas. Estamos contentos con el resultado, y hemos aprendido a hacer un sistema complejo de reservas con varias lógicas de horarios, precios, etc. Si hubiésemos seguido la primera idea del TFG hubiera surgido una red social destinada al deporte, que de hecho es una forma de mejorar la aplicación y comentaremos en el siguiente apartado. Aquí nos gustaría añadir que hemos aprendido a gestionar un proyecto, hemos visto que, si no se definen bien el producto a desarrollar, este puede verse deteriorado durante el desarrollo (cosa que por suerte no nos ha ocurrido aquí, simplemente hemos sacado otro producto, que puede ir de la mano perfectamente con el original). Hemos aprendido a desarrollar una aplicación en Android que sea capaz de comunicarse con este sistema y cómo puede construirse un producto realmente atractivo. Hemos usado herramientas como ***Trello, Jira*** y ***Github*** para realizar todo el proyecto y ha dado como resultado lo que creemos que es un producto bastante profesional.

4.2. Posibles Mejoras

Creo que la forma más lógica de mejorar este producto es implementar la idea original, y es creando una especie de red social, pero sin llegar a convertirnos a en eso. Simplemente crear un lugar donde los usuarios puedan encontrar a otros usuarios para realizar el deporte; esto se puede conseguir fácilmente creando chats, ligas, torneos, reservas conjuntas, etc. Un ejemplo de eso sería indicar cuántas personas han reservado una pista y publicar las plazas restantes para que se puedan unir los demás usuarios.

REFERENCIAS BIBLIOGRÁFICAS

http://laravel.com/ → Sitio web del Framework Laravel donde hemos consultado toda la información para su uso.

https://www.flickr.com/ → Sitio web donde se almacenan millones de imágenes algunas de propósito libre, estas imágenes han sido usadas para publicarlas en nuestra aplicación web.

http://stackoverflow.com/ → Foro web destinado a desarrolladores donde se comparten conocimientos y soluciones a problemas, hemos usado esta web para consultar algunos errores encontrado durante el desarrollo.

http://sass-lang.com/ → Documentación oficial del lenguaje de programación de estilos SASS, usado para compilar todos los CSS de la versión web.

http://jquery.com/ → Documentación oficial para el Framework de JavaScript JQuery, usado para algunas animaciones y peticiones Ajax dentro de la versión web.

http://getbootstrap.com/ → Documentación oficial del Framework CSS Bootstrap, desarrollado por Twitter y usado como base de la versión web.

http://github.com/ → Documentación oficial de muchos de los paquetes de PHP usados para el desarrollo del TFG.

https://packagist.org/ → Documentación oficial de muchos de los paquetes de PHP usados para el desarrollo del TFG.

ANEXOS TÉCNICOS

I. Manual de instalación

I.I. SISTEMA OPERATIVO

Como sistema operativo hemos usado un Ubuntu Server 16.04.2 LTS. Aunque añadiremos la máquina virtual totalmente operativa con todo el servicio, vamos a realizar un pequeño manual de instalación por si se desea usar en otras condiciones.

I.II. PAQUETES INSTALADOS

Vamos a hacer un resumen de todos los paquetes necesarios para arrancar el sistema, si es necesario, explicaremos algunos de ellos:

- ***git-man***
- ***git***
- ***php-common***
- ***php-imagick***
- ***libxrender1***
- ***php7.0-common***
- ***php7.0-json***
- ***php7.0-opcache***
- ***php7.0-readline***
- ***php7.0-cli***
- ***php7.0-fpm***
- ***php7.0-mysql***
- ***php7.0-mbstring***
- ***php7.0-bcmath***
- ***php7.0-gd***
- ***php7.0-xml***
- ***php7.0-zip***
- ***php7.0***
- ***nginx*** (y por supuesto todas las diferencias)
- ***mysql-server***
- ***libzip4***
- ***unzip***

I.III. BASE DE DATOS

Para la base de datos hemos usado ***MySQL 5.7.18*** y hemos creado una tabla usando el comando:

CREATE DATABASE fit_and_meet CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;

Con este comando ya tendremos una base de datos donde vamos almacenar todos los datos de la aplicación.

I.IV. GESTOR DE PAQUETES

Ahora procedemos a instalar ***composer*** que es un gestor de paquetes en PHP necesario para descargar todas las librerías usadas en Laravel. Para instalar ***composer*** es tan sencillo como seguir los pasos que aparecen en la web oficial. Una vez instalado correctamente ejecutamos en la raíz del directorio el comando:

composer update

De esta forma se instalarán automáticamente todos los paquetes necesarios para ejecutar la aplicación.

I.V. LARAVEL

Lo siguiente es preparar el proyecto, para ello copiamos el archivo ***.env.example*** con el nombre de ***.env***. Una vez que tengamos este fichero ejecutaremos el comando:

php artisan key:generate

Este comando genera una clave de seguridad para nuestra aplicación. Una vez generada la clave, establecemos la URL de la aplicación, los datos de la base de datos creada anteriormente y los datos del servidor de correo.

Acto seguido ya podemos ejecutar también el comando:

php artisan migrate –seed

Este comando básicamente ejecuta todas las migraciones de nuestra aplicación creando todas las tablas, y gracias a la opción ***–seed*** podemos además añadir algunos datos de prueba; si no queremos esto último, basta con quitar esta opción.

I.VI. H4CC

Antes de poder ejecutar la aplicación correctamente, tenemos que hacer un par de cambios más: copiaremos dos ficheros a una localización deseada, los archivos son ***wkhtmltoimage-amd64*** y ***wkhtmltopdf-amd64*** vamos a copiarlos al directorio ***/usr/local/bin*** en este caso, aunque se puede cambiar la localización cambiándola en el fichero ***config/snappy.php***. Los ficheros de origen se encuentran en:

- ***vendor/h4cc/wkhtmltoimage-amd64/bin/wkhtmltoimage-amd64***
- ***vendor/h4cc/wkhtmltopdf-amd64/bin/wkhtmltopdf-amd64***

Cuando copiemos los ficheros en la carpeta de destino tendremos que darles a los ficheros los permisos de ejecución.

Con esto ya estaría todo listo para el uso normal de la aplicación, a excepción de la API.

I.VII. API

Por último, tenemos la parte de la API, esta es la más sencilla de preparar. Para ello primero tenemos que ejecutar el siguiente comando:

php artisan passport:keys

Con este comando se generan unas claves con las que se generarán todos los tokens de autenticación, y demás funcionamiento de la API.

Lo siguiente que tenemos que hacer es que, por cada cliente que queramos diferenciar, tendremos que ejecutar el siguiente comando. Por ejemplo, en nuestro caso queremos dos clientes bien diferenciados, uno para Android y otro para iOS, así que lo ejecutaremos dos veces:

php artisan passport:client –password

Esto generará un cliente para una de las plataformas, devolviendo un ***ID*** y un ***SECRET***, esto se guardará en la base de datos, pero podríamos tenerlo documentado en algún lado ya que son necesarios para poder usar la API desde las aplicaciones.

Con esto ya tendremos la API lista para funcionar.