Universitat Oberta de Catalunya

uoc.edu

Aplicación T-Golf

Gorka Estévez González
Grado de Ingeniería Informática
Desarrollo Web

Gregorio Robles Martínez
Santi Caballe Llobet

09/01/2020

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Aplicación T-Golf*
Nombre del autor:	*Gorka Estévez González*
Nombre del consultor/a:	*Gregorio Robles Martínez*
Nombre del PRA:	*Santi Caballe Llobet*
Fecha de entrega (mm/aaaa):	01/2020
Titulación:	*Grado de Ingeniería Informática*
Área del Trabajo Final:	*Desarrollo Web*
Idioma del trabajo:	*Español*
Palabras clave	*Desarrollo web, Node, JavaScript*

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados y conclusiones del trabajo.*

Este documento describe el trabajo de desarrollo de una aplicación web para la gestión eficiente de los campos de golf, T-Golf. La metodología utilizada divide la aplicación en dos partes: *backend* y *frontend*. El *backend* se desarrolla con Node, Express y mongoDB; el *frontend*, con HTML5, CSS, jQuery, Bootstrap y Angular. El resultado es una aplicación que gestiona los datos recibidos por los diferentes dispositivos presentes en el campo, que administra en tiempo real la base de datos, muestra la información requerida por los usuarios del sistema de una forma clara y sencilla, siendo accesible desde cualquier lugar gracias a Internet.

Este desarrollo del proyecto me ha permitido elaborar una aplicación desde cero aprendiendo en el proceso todos los lenguajes y herramientas necesarias.

Abstract (in English, 250 words or less):

This document describes the development of a web application (T-Golf) for the efficient management of golf courses. The methodology used divides the application into two parts: backend and frontend. The backend has been developed with Node, Express and mongoDB; on the other hand, the frontend has been developed with HTML5, CSS, jQuery, Bootstrap and Angular. The result is an application that manages the data received from the different devices on the courses, manages the database in real-time, shows the information required by the users of the system clearly and simply, and can be accessed from anywhere thanks to the Internet.

The development of this project has allowed me to create an application from scratch, learning in the process of all the necessary languages and tools.

Índice

Lista de figuras

1. Introducción

1.1 Contexto y justificación del Trabajo

TAFCO Metawireless es una pyme cuya actividad es la de proveer tecnología y consultoría a empres o instituciones en el sector del diseño, análisis, implantación y mantenimiento de productos y soluciones de telecomunicación y telemedida. Ha sido reconocida como EIBT nacional y ganadora del premio Gaztempresa al mejor proyecto empresarial. Entre los numerosos proyectos que tiene la TAFCO, actualmente se encuentra desarrollando un sistema para la gestión eficiente de los campos de golf (T-Golf) en un entorno preindustrializable para pruebas académicas. Y es sobre este entorno que se desarrollará el proyecto que nos ocupa.

El proyecto que se desarrollará en este Trabajo de Fin de Grado consistirá en diseñar, desarrollar e implantar la capa de aplicación para un sistema piloto de la gestión eficiente de los campos de golf, apoyado en el entorno preindustrializable mencionado en el párrafo anterior.

Teniendo en cuenta que actualmente existen 359 campos de golf en España y que es el segundo país del mundo en la recepción del turismo relacionado con el golf, parece necesaria una modernización en los sistemas de gestión del juego y/o campos. Hoy en día la gestión del juego se lleva a cabo mediante la “buena fe” de los jugadores y el criterio subjetivo del caddie master. Esta situación hace que en ocasiones surjan disputas entre los jugadores entorpeciendo el correcto desarrollo del juego. Para solucionar este tipo de problemas o simplemente proveer de información a los usuarios de los campos de golf, desde gerentes hasta jugadores, nace la idea de proporcionar una aplicación mediante la cual se puedan obtener los datos y herramientas necesarias para una correcta gestión del juego.

Por lo tanto, la aplicación que se desarrollará será capaz de gestionar todos los datos recogidos por los dispositivos presentes en el campo en tiempo real, ordenarlos en una base de datos y permitir su visualización en diferentes tipos de dispositivos, como teléfonos inteligentes, tabletas o PC. Los datos obtenidos y mostrados servirán para evitar problemas de “juego lento”, muy frecuentes en los campos de golf, así como para dar información relevante a los jugadores, o a los responsables del campo para una rápida y eficiente toma de decisiones.

1.2 Objetivos del Trabajo

Como ya se ha comentado, el objetivo principal de este trabajo es de desarrollar una aplicación para la gestión del juego de los campos de golf. Esta aplicación, a su vez, deberá cumplir los siguientes objetivos:

- Gestionar los datos recibidos por los diferentes dispositivos presentes en el campo.
- Administrar en tiempo real la base de datos.
- Procesar los datos recogidos de manera que estos puedan ser de utilidad para los usuarios del sistema.
- Implantar medidas de seguridad para evitar usos malintencionados.
- Visualizar de forma clara y sencilla las diferentes opciones que permite el sistema a través de diferentes dispositivos.
- Hacer la aplicación accesible desde cualquier lugar gracias a Internet.

1.3 Enfoque y método seguido

Para llevar a cabo el trabajo se dividirá el desarrollo en dos partes, por un lado, se creará el *backend* o programación del lado del servidor que pertenece al nivel 2; y, por otro lado, el *frontend* que pertenece al nivel 3 o nivel de visualización.

Para desarrollar el *backend* se hará uso de:

- Node.js: un entorno de ejecución JavaScript multiplataforma orientado a eventos asíncronos y está diseñado para la creación de aplicaciones de red escalables. Además, está diseñado para optimizar el rendimiento y escalabilidad en aplicaciones web, está escrito en Simple JavaScript (que se beneficia de los diferentes avances en diseño de lenguajes), dispone de cientos de paquetes reutilizables, es portable y posee una comunidad de desarrolladores muy activa.

- MongoDB: sistema de gestión de bases de datos NoSQL distribuido de tipo documental, escrito en C++, multiplataforma y de código abierto. Se opta por un sistema de gestión de bases de datos NoSQL ya que estos permiten flexibilidad en la definición de los datos, sencillez de acceso a estos, rendimiento elevado y gran escalabilidad. Además, las bases de datos documentales son especialmente adecuadas para crear aplicaciones web.

La parte del *frontend* se desarrollará con las siguientes herramientas:

- HTML5: quinta revisión del lenguaje básico de la World Wide Web, en la que se desarrollará principalmente la aplicación.
- CSS: lenguaje de diseño gráfico que permite definir y crear la presentación de un documento escrito en algún lenguaje de marcado. Es muy usado para establecer la hoja de estilo de los sitios web y/o interfaces de usuario escritas en HTML o XHTML. Junto a HTML y JavaScript es muy popular entre los desarrolladores de aplicaciones web.
- jQuery: es una biblioteca de JavaScript que permite simplificar la manera de interactuar con documentos HTML, manejar eventos, manipular el árbol DOM (*Document Object Model*), desarrollar animaciones y agregar interacción con la técnica AJAX.
- Bootstrap: *framework* CSS y JavaScript diseñado para la creación de interfaces limpias y con un diseño adaptable. Actualmente, es una de las herramientas más utilizadas para diseñar sitios web como aplicaciones, ya que permite que las creaciones sean 100% adaptables a cualquier tipo de dispositivo.

La aplicación se deberá enfocar para dar soporte a diferentes tipos de usuarios, como, por ejemplo: los administradores de la aplicación, jugadores o caddie master.

1.4 Planificación del Trabajo

Para el desarrollo de la aplicación web se hará uso del editor de texto Atom que permite gran flexibilidad a la hora de programar en gran cantidad de lenguajes como pueden ser Node o HTML. Además, será necesario un servidor donde alojar la base de datos MongoDB, así como el servidor web de la aplicación. Por último, pero no menos importante, serán necesarios los dispositivos de recolección de datos del juego proporcionados por la empresa Tafco Metawireless para las pruebas en un entorno real de la aplicación.

Para la realización del proyecto se han contemplado cuatro grandes hitos, que se indican a continuación:

- Hito 1 – Especificación de necesidades: en este primer hito se recogerán los requisitos que debe cumplir la aplicación. Estos requisitos provendrán de un estudio de las necesidades de los futuros usuarios de la aplicación.
- Hito 2 – Diseño de la aplicación y la relaciones con los diferentes componentes: en el segundo hito se diseñará la aplicación (cómo debe funcionar, qué pantallas tendrá, etc.) y se detallarán las relaciones entre los diferentes componentes de esta (base de datos, página web, etc.).
- Hito 3 – Programación de la aplicación y pruebas en entorno preproductivo: en este hito se programará la aplicación y se creará una maqueta usable y lo más fiel posible al resultado final esperado.

- Hito 4 – Realización de la memoria final del trabajo: en el último hito se desarrollará la memoria final del TFG, además de una presentación en video que sintetizará el trabajo realizado durante el semestre.

El siguiente diagrama de Gantt nos muestra la planificación temporal:

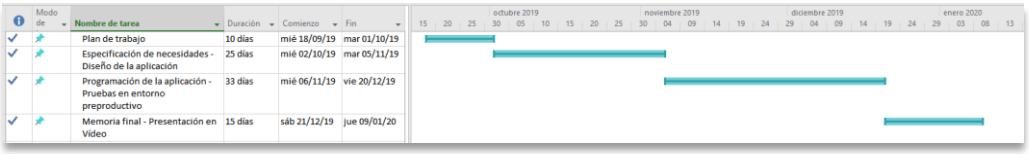


Figura 1. Planificación temporal. Diagrama de Gantt

1.5 Breve sumario de productos obtenidos

Los resultados, o productos obtenidos, de los diferentes hitos del trabajo serían los siguientes:

- Documento con todas las necesidades que tiene que cubrir la aplicación.
- Documento con el diseño de lo que se tiene que programar.
- Código fuente de la aplicación.
- Maqueta funcional para poder usarse en un entorno de pruebas o preproductivo.
- Memoria final del TFG.
- Presentación en vídeo del TFG.

1.6 Breve descripción de los otros capítulos de la memoria

Esta memoria se compone de nueve capítulos contando el de la introducción:

- En el segundo capítulo, Análisis y diseño del sistema, se especifica qué va a necesitar la aplicación y cómo se va a diseñar para obtener los objetivos marcados.
- El tercer capítulo, Tecnologías y lenguajes, se explican las tecnologías y lenguajes utilizados para desarrollar la aplicación.
- En el cuarto capítulo, Implementación, se detalla como se ha programada la aplicación, así como la estructura de archivos utilizada.
- El quinto capítulo, Resultado, muestra un resumen de la consecución del objetivo explicando las funcionalidades del sistema.
- En el capítulo sexto, Conclusiones, se resumen las conclusiones a las que se ha llegado tras la finalización del Trabajo de Fin de Grado.
- Los capítulos siete, ocho y nueve, Glosario, Bibliografía y Anexos, se explican por sí solos.

2. Análisis y diseño del sistema

2.1 Introducción

La plataforma T-Golf está compuesta por una capa física de dispositivos que se encargan de recopilar los datos de los jugadores y una capa de aplicación que permite la gestión y visualización de toda la información recolectada por los dispositivos que conforman la capa física.

Aunque el presente proyecto se centra únicamente en el desarrollo de la capa de aplicación, a continuación, se realiza una pequeña introducción de los diferentes dispositivos que conformarán la capa física de la plataforma. De este modo, el lector tendrá una visión global del funcionamiento de la plataforma completa.

Los End-Devices serán los encargados de recoger la información que será procesada, y por lo tanto, son la base de todo el sistema. Se diferencian cinco tipos principales por sus características particulares: los User-ED, UserM-ED, Sensor-ED, SensorW-ED y Flag-ED.

Los User-ED serán dispositivos-sensóricos dotados con la capacidad de estimar la posición del jugador. A partir de esta información, se harán cálculos de diferente índole para determinar, por un lado, la velocidad de juego y por otro la distancia al centro, inicio y final del *green*. Además, ofrecerá al usuario la información de distancias junto con el número de hoyo en el que está jugando. Adicionalmente, posibilitará otro tipo de información, como avisos del club.

Figura 2. Prototipo del User-ED de la plataforma

Los UserM-ED serán dispositivos-sesóricos similares a los User-ED, pero dirigidos a la gestión de recursos materiales y humanos del área de mantenimiento. Los dispositivos estarán dotados con la capacidad de estimar la posición del recurso de mantenimiento a través de un receptor GPS. A partir de esta información, se podrán analizar, por ejemplo, las rutas empleadas por las máquinas en sus tareas de jardinería, la localización de los equipos a lo largo de la jornada, etc. Además, el dispositivo contará con una pantalla donde se podrá mostrar información al responsable de mantenimiento.

Los Sensor-ED se utilizarán fundamentalmente para monitorizar el estado del terreno, por lo que irán soterrados. En cada *green* se situará uno, ya que estos necesitan de un cuidado especial y para no influir en la dinámica de juego no podrán tener ninguna parte en el exterior. La duración de su batería será muy prolongada, evitando su levantado.

Los SensorW-ED captores de información medioambiental, al ser dispositivos que no tienen que ser transportados por los jugadores; los requisitos de miniaturización son más laxos, por lo que se analizará la posibilidad de utilizar soluciones de “energy harvesting” o energía verde; por ejemplo, alimentando los dispositivos con placas solares.

Por último, los Flag-ED serán sensores insertados en los hoyos para detectar si la bandera está situada en su sitio. El hecho de implementar este sensor responde a la necesidad de determinar cuándo se ha empezado y terminado el juego en el *green*; ya que siempre que se juega en esta zona, se quita la bandera y al acabar se vuelve a colocar. Esta información se complementará con la obtenida por los User-ED, para la determinación de los tiempos intermedios de cada hoyo.

La capa de aplicación será capaz de gestionar los datos recogidos por todos los dispositivos y mostrarlos de una forma ordenada, cuando estos sean requeridos. A partir de esta información, se podrán obtener las estadísticas que determinan si el juego está siendo lento, aportando información útil al club para tomar medidas y un valor añadido a los jugadores al permitir consultar, por ejemplo, la distancia a cada uno de los hoyos del campo o el tiempo que les ha costado realizarlos. Asimismo, se podrán conocer las condiciones del terreno de forma sencilla, posibilitando la activación de los mecanismos oportunos de irrigación y cuidado del césped en general.

La capa de aplicación es un punto fundamental de la plataforma, ya que es necesario implementar un software que, en tiempo real para el usuario, sea capaz de recoger centenares de datos de diferentes dispositivos, gestionarlos en una base de datos y visualizarlos en diferentes tipos de dispositivos, como Smartphones, Tablets o PCs.

La aplicación que se desarrollará ofrecerá una solución *multi-tenant* para que sobre una misma plataforma las diferentes empresas encargadas de la gestión de los campos de golf puedan gestionarlo de manera independiente. Por descontado, esta solución *multi-tenant* ha de ser totalmente transparente a las diferentes empresas que hagan uso de la plataforma. Además, de poder dar

servicio a diferentes empresas, la aplicación deberá de ser capaz de gestionar diferentes perfiles de usuario para limitar las acciones y visualizaciones en función de las “habilidades” o permisos de cada usuario.

A grandes rasgos, el sistema funcionará de la siguiente manera: los datos obtenidos por los diferentes sensores presentes en el campo (ya sean fijos o los que portan los jugadores) serán enviados a un servidor que estará conectado a una base de datos donde se guardará toda la información recibida para poder ser tratada. Este servidor central, contará a su vez con un servidor web para que los usuarios de la plataforma puedan interactuar con el sistema.

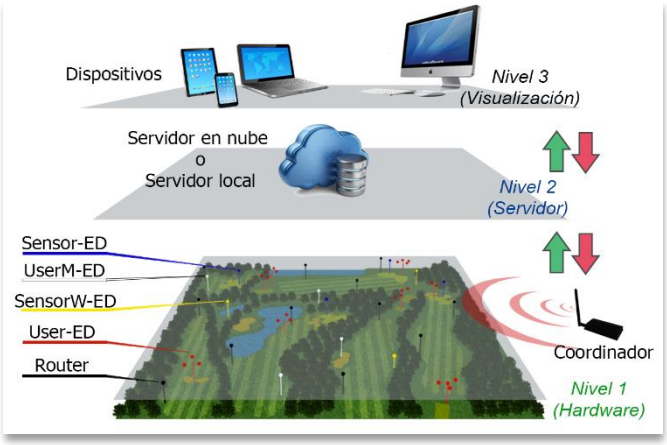


Figura 3. Esquema de funcionamiento de la plataforma

2.2 Especificaciones *multi-tenant*

La solución *multi-tenant* ofrecida por la plataforma tendrá tres unidades diferenciadas que se explican a continuación:

- Tenant Default: o Tenant 0, es el *tenant* por defecto para la administración de la plataforma. Aquí se podrán configurar los parámetros generales de la plataforma y se administrarán los diferentes *tenants*.
- Tenant: unidad que hace referencia a cada una de las entidades que se encargan de gestionar los campos de golf. Estas entidades podrán gestionar uno o varios campos.
- Campo: unidad organizativa que utilizaremos dentro de cada *tenant* para identificar un terreno de juego y englobar a los diferentes dispositivos que se encuentran en este.

A continuación, pasamos a describir los datos necesarios para los *tenants* y los campos:

- Tenant (colección en MongoDB, *tenants)*:
  - tenantid*: clave única para identificar el *tenant* dentro de la plataforma.
  - cif*: código de empresa, este código es único.
  - name: nombre de la empresa.
  - address: dirección fiscal de la empresa.
  - phone*: teléfono de contacto.
  - email*: dirección de correo, la dirección será única dentro del sistema.

- Campo (colección en MongoDB, *courses*):
  - courseid*: clave única para identificar el campo dentro de la plataforma.
  - tenantid*: identificador del *tenant* al que pertenece el campo.
  - address: dirección física del campo, si no se especifica la dirección se usa la del *tenant* por defecto.
  - phone: teléfono de contacto, si no se especifica el número de teléfono se usa el del *tenant* por defecto.
  - email: dirección de correo, si no se especifica se usa la dirección de correo del *tenant*.
  - holes*: número de hoyos del campo.
  - longitude*: necesario para ubicar el campo en la vista de mapa.
  - latitude*: necesario para ubicar el campo en la vista de mapa.

- Hoyo (colección en MongoDB, *holes*):
  - holeid*: clave única para identificar el hoyo dentro de la plataforma.
  - courseid*: identificador del campo al que pertenece el hoyo.
  - holenr: número del hoyo dentro del campo.
  - par*: número estipulado de golpes para acabar el hoyo.
  - length*: distancia desde la salida del hoyo hasta la bandera.
  - longitude*: necesario para ubicar el hoyo en la vista de mapa.
  - latitude*: necesario para ubicar el hoyo en la vista de mapa.

2.3 Especificaciones de los Perfiles

Para poder acceder a la plataforma es indispensable estar autorizado, y serán los permisos o “habilidades” de los usuarios los que marquen a que funcionalidades del sistema se podrá acceder. Estos niveles de acceso componen un sistema jerárquico, siendo el nivel más bajo el que tenga menos privilegios y por tanto acceso a menos funcionalidades. Los niveles superiores tendrán acceso, como mínimo, a las mismas funcionalidades que el nivel o niveles que tengan por debajo, a excepción de las funcionalidades específicas del caddie master y de los jugadores.

El sistema tiene definidos cinco perfiles distintos:

- Perfil Administrador: privilegios totales para acceder y visualizar todas las características de la plataforma. Los usuarios con este perfil serán los únicos capaces de dar de alta nuevos *tenants* y nuevos usuarios administradores del sistema. Será el único usuario capaz de eliminar o editar cualquier usuario del sistema.
- Perfil Admin Tenant: privilegios únicamente sobre su *tenant*. Estos usuarios serán los encargados de la administración de un *tenant* en concreto. Su nivel de acceso les permitirá:
  - Modificar los parámetros particulares de su *tenant*.
  - Dar de alta nuevos campos y gestionar (editar o borrar) los existentes.
  - Generar usuarios de menor privilegio al suyo dentro de su *tenant* como caddie masters o técnicos de mantenimiento.
  - Editar o eliminar los usuarios que pertenecen a su tenant.
- Perfil Caddie Master: el caddie master es la persona encargada de coordinar los caddies y asignarlos a los jugadores, pero en realidad tienen más responsabilidades ya que en la mayoría de los campos de golf no hay caddies. Su nivel de acceso le permitirá:
  - Visualizar pestaña de caddie master para el control de juego. En esta vista el caddie master tendrá una visión global de los jugadores participantes en el juego en tiempo real en cualquiera de los campos del tenant.
  - Generar usuarios de tipo jugadores, pero no podrá editarlos ni eliminarlos
- Perfil Técnico Mantenimiento: personal encargado del correcto funcionamiento de los equipos de monitorización.
  - Puede visualizar la pestaña de Mantenimiento para el control de alarmas y eventos. Esta vista le permitirá además de comprobar los logs de los equipos de monitorización comprobar su situación en tiempo real dentro de los campos del tenant al que pertenece.
  - Dar de alta y gestionar (editar o borrar) los equipos de monitorización.
- Perfil Jugador: los usuarios básicos del campo de golf.
  - Puede visualizar la pestaña de Jugador.
  - Puede modificar los parámetros básicos de su perfil.

Los datos necesarios para los usuarios del sistema serán (colección en MongoDB, *users*):

- username*: nombre único en el sistema que servirá para identificar al usuario de forma unívoca.
- tenant*: ID del *tenant* al que pertenece el usuario. Por defecto toma el valor 99999 que es el tenant comodín para los jugadores.
- name*: nombre del usuario.
- password*: contraseña del usuario.
- email*: correo electrónico del usuario. Debe de ser único en el sistema.
- permission*: nivel del usuario dentro del sistema. Su valor por defecto es 'user', el nivel más bajo.
- address: dirección de contacto.
- phone: número de teléfono de contacto.
- active*: indica si el usuario se ha activado. Su valor por defecto es 'falso'.
- temporarytoken*: token que permite activar al usuario. Cuando se activa un usuario, su valor pasa a 'falso'.
- resettoken: token que se utiliza para reiniciar la contraseña del usuario.

Dado que el sistema ha de proporcionar información acerca del juego para una correcta gestión de este por parte del caddie master y como información de valor añadido a los jugadores, se deberán guardar los datos del juego recogidos por los sensores en la base de datos. Además de estos datos, la información de los sensores repartidos por los campos también se guardará en la base de datos para que pueda ser accesible por parte de los técnicos de mantenimiento.

La información del juego activo que se guardará será la siguiente:

- Juego Activo (colección en MongoDB, *trackers*):
  - trackerid: clave única para identificar el dispositivo GPS (rastreador) dentro del sistema.
  - courseid: ID del campo asociado al juego.
  - playerid: ID del jugador que está jugando.
  - inuse: valor que indica si el rastreador se está utilizando.
  - longitude: necesario para ubicar el jugador en la vista de mapa.
  - latitude: necesario para ubicar el jugador en la vista de mapa.

La información de los partidos de los jugadores será la siguiente:

- Juego (colección en MongoDB, *games*):
  - gameid: clave única para identificar el juego dentro del sistema.
  - courseid: ID del campo asociado al juego.
  - username: nombre de usuario del jugador asociado al juego.
  - holenr: número de hoyos del juego.
  - coursepar: par del campo donde se ha desarrollado el partido.
  - score: resultado del partido (número de golpes para completar el recorrido).
  - date: fecha de realización del partido.

- Detalles del juego (colección en MongoDB, *gameholes*):
  - gameholeid: clave única para identificar el hoyo dentro del sistema.
  - gameid: ID del juego asociado al hoyo.
  - hole: número del hoyo en el campo donde se ha desarrollado el juego.
  - holepar: par del hoyo.
  - score: resultado del hoyo (número de golpes para completar el hoyo).

La información de los dispositivos de monitorización que se guardará será la siguiente:

- Dispositivo (colección en MongoDB, *devices*):
  - deviceid: clave única para identificar el dispositivo dentro del sistema.
  - courseid: ID del campo donde se encuentra el dispositivo.
  - type: tipo de dispositivo según su función.
  - status: estado de funcionamiento del dispositivo.
  - info: mensaje relacionado con el estado del dispositivo.
  - longitude: necesario para ubicar el dispositivo en el campo.
  - latitude: necesario para ubicar el dispositivo en el campo.

- Información del dispositivo (colección en MongoDB, *deviceInfo*):
  - deviceid: ID del dispositivo asociado a la información.
  - devicetype: tipo de dispositivo según su función.
  - message: mensaje relacionado con el estado del dispositivo.
  - date: día y hora del mensaje.

A continuación, se muestra esquemáticamente la relación entre las diferentes colecciones de la base de datos de la aplicación:

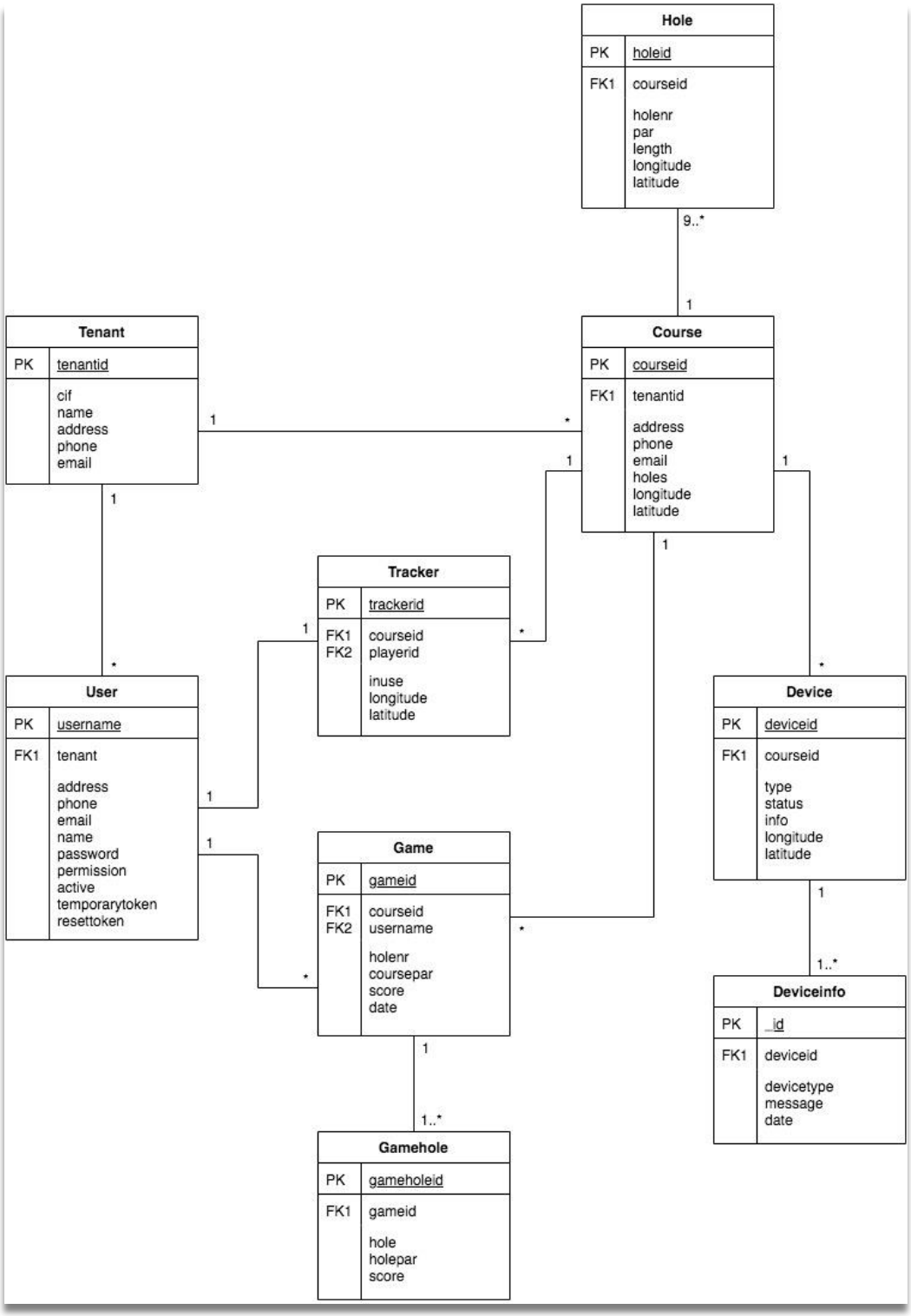


Figura 4. Diagrama Entidad-Relación

2.4 Diseño

2.4.1 Consideraciones previas

Antes de pasar a explicar cómo se ha implementado la aplicación T-Golf, es necesario detallar el equipo y los programas y servicios que se han utilizado para el desarrollo del proyecto.

La aplicación se ha desarrollado en un equipo con las siguientes características:

- Sistema Operativo: Windows 10 Pro
- Procesador: Intel® Core™ i7-3820 CPU @ 3.60GHz
- Memoria RAM: 16 GB
- Tipo de sistema: sistema operativo de 64 bits
- Tarjeta Gráfica: Radeon RX 570 Series
- Memoria RAM de vídeo: 4GB GDDR5

Para poder programar y testear la aplicación se han utilizado los siguientes programas y servicios:

- NodeJS
- Atom
- MongoDB Atlas
- Twilio SendGrid
- Diferentes navegadores web: Chrome, Mozilla Firefox, Safari y Edge

En primer lugar, será necesario instalar NodeJS. Para desarrollar la aplicación se ha utilizado la versión 10.16.3 (LTS) aunque actualmente la versión LTS es la 12.14.0. Para instalar Node se puede acceder a la página https://nodejs.org/ y descargar la versión LTS o la más actual para el sistema operativo que se desee.

Para escribir el código de la aplicación se puede utilizar cualquier editor de texto plano o plataforma de programación. En el caso que nos ocupa se ha escrito la aplicación utilizando el editor de texto Atom, el cual se puede descargar desde https://atom.io/.

Para la creación y gestión de la base de datos que dará soporte a la aplicación se ha optado por usar el servicio en la nube que ofrece MongoDB, MongoDB Atlas. Este servicio dispone de un plan gratuito que ofrece 512 MB de espacio de almacenamiento y RAM compartida en un clúster alojado en aws, Google Cloud Platform o Azure. Aunque esta configuración se queda corta en un entorno productivo, es más que suficiente para el desarrollo de la aplicación y las pruebas que se quieran realizar.

Por último, se ha utilizado la plataforma SendGrid para hacer los envíos de correo electrónico desde la aplicación. Esta plataforma en la nube ofrece un plan gratuito de 100 correos al día que es el que se ha usado en la aplicación.

2.4.2 Estructura de la aplicación

La aplicación se ha estructurado en dos grandes grupos, por un lado, estaría la parte del *frontend*, que sería la parte con la que interactúan los usuarios, y por otro la parte de *backend*, que sería la parte donde se ejecutan las acciones solicitadas por los usuarios y que es transparente para estos.

A continuación, se resume la estructura, que se desarrollará más en detalle en los siguientes apartados:

- app: es la carpeta que hace referencia a la parte de *backend* de la aplicación.
  - models: contiene todos los modelos utilizados por todas las colecciones de la base de datos de MongoDB.
  - routes: contiene el núcleo de la aplicación. Aquí se definen todas las llamadas HTTP que se realizan para la comunicación con la base de datos.

- node_modules: carpeta que contiene todas las librerías y módulos externos necesarios de Node.

- public: esta carpeta hace referencia a la parte de *frontend* de la aplicación.
  - app:
    - controllers: carpeta que contiene los controladores necesarios para la comunicación entre las vistas y el modelo.
    - services: contiene la definición de los servicios para llamar a las funciones que se ejecutarán en la parte de *backend*.
    - views: contiene todas las vistas (páginas) de la aplicación.
    - app.js: este archivo es el encargado de cargar todos los controladores y servicios.
    - routes.js: contiene todas las rutas de la aplicación web y sus características.

  - assets: contiene “activos” de la aplicación. Como imágenes, archivos de estilos, scripts, etc.

- package-lock.json: archivo que se genera de forma automática cuando se modifica el árbol de carpetas de la carpeta *node_modules* o el archivo *package.json*.
- package.json: archivo con la configuración del proyecto.
- server.js: este es el archivo que arranca la aplicación web.

2.4.3 Modelos de la base de datos

En la carpeta *models*, se encuentran todas las definiciones de los esquemas de las colecciones definidas para la aplicación. MongoDB usa colecciones para almacenar múltiples documentos, los cuales no tienen por qué tener la misma estructura, aunque cuando se trata con objetos es necesario que los documentos sean similares.

Se utiliza la librería *mongoose* para realizar la conexión con la base de datos, así como la creación de los esquemas para los diferentes modelos.

Mongoose usa un objeto "Schema" para definir una lista de propiedades del documento (campos), cada una con su propio tipo y características para forzar la estructura del documento.

Para crear un *schema* para definir una colección de documentos se utiliza el siguiente código:

```
var mongoose   = require('mongoose');
var Schema     = mongoose.Schema;
var validate   = require('mongoose-validator');

//useCreateIndex avoids warning
mongoose.set('useCreateIndex', true);

var deviceidValidator = [
  validate({
    validator: 'isLength',
    arguments: [5],
    message: 'Device ID must be {ARGS[0]} numbers long.'
  }),
  validate({
    validator: 'matches',
    arguments: /^[0-9]*$/,
    message: 'Device ID must be an integer number.'
  })
];

var DeviceSchema = new Schema({
  deviceid: { type: String, required: true, unique: true, validate: deviceidValidator },
  courseid: { type: String, required: true },
  type: { type: String, required: true },
  longitude: { type: String },
  latitude: { type: String },
  status: { type: String },
  info: { type: String }
});

module.exports = mongoose.model('Device', DeviceSchema, 'devices');
```

Figura 5. Ejemplo de creación de esquema y colección

En el código anterior se realizan dos acciones: se define el objeto *DeviceSchema* usando el constructor *Schema*, y luego se usa la instancia del esquema para definir el modelo *Device*. Por defecto, al crear el modelo, *mongoose* utilizará el plural del nombre del modelo en minúscula, *devices* en este caso, para crear la colección en la base de datos. A la hora de programar la aplicación siempre se ha especificado el nombre de la colección.

Después de especificar un esquema se deberá definir un Modelo constructor que se usará para crear instancias de los documentos de MongoDB. Así, de esta manera se pueden crear documentos añadiendo objetos que tengan la estructura del esquema que hemos definido. Por ejemplo:

```
var Device      = require('../models/device');
var device      = new Device(req.body);
```

Donde *req.body* será un objeto con una estructura que coincida con el esquema definido para *Device*.

Para la aplicación se ha creado la siguiente estructura de archivos donde se definen diferentes esquemas que construyen una serie de colecciones que guardan todos los datos necesarios del sistema. A continuación, se define la organización de archivos dentro de la carpeta *models*:

- course.js: contiene la información de los campos de golf del sistema
- device.js: contiene la información básica de los dispositivos del sistema
- deviceInfo.js: contiene la información de los mensajes que generan los dispositivos.
- game.js: contiene la información básica de los partidos.
- gamehole.js: contiene la información detallada de cada partido.
- hole.js: contiene la información de todos los hoyos de los diferentes campos del sistema.
- tenant.js: contiene la información de las empresas del sistema.
- tracker.js: contiene la información de los dispositivos de seguimiento de los jugadores.
- user.js: contienen la información de todos los usuarios del sistema.

2.4.3.1 tenant.js

Este archivo contiene el esquema para crear la colección donde se guardarán los diferentes *tenant* de la aplicación. Los documentos que se guardarán en la colección tienen los siguientes campos:

- tenantid: identificador único para cada *tenant*.
  - type: String
  - required: true
  - unique: true
  - validate: tenantidValidator

Este campo siempre tiene que estar presente a la hora de guardar un nuevo *tenant* en la colección y su valor tiene que ser único. No se pueden guardar dos o más *tenant* con el mismo identificador. Además, este campo tiene que pasar un proceso de validación antes de que el nuevo documento se guarde satisfactoriamente. En este caso el *backend* comprueba que el campo tenga siempre 5 caracteres y que esos caracteres estén entre 0 y 9.

- cif: código de identificación fiscal de la empresa.
  - type: String
  - required: true
  - unique: true
  - validate: cifValidator

Campo con las mismas características que el anterior. Es decir, es obligatorio y debe de ser único. Además, pasa por un proceso de validación que comprueba que el campo tenga siempre 9 caracteres y que sea una cadena alfanumérica.

- name: nombre de la empresa.
  - type: String

- email: correo electrónico de la empresa.
  - type: String
  - lowercase: true
  - required: true
  - unique: true
  - validate: emailValidator

Este campo tiene unas características similares a los campos *tenantid* y *cif*. Es obligatorio y debe de ser único, pero, además, tienen que estar escrito en minúsculas. Al igual que los otros dos campos, pasa por un proceso de validación que comprueba que la longitud de la cadena de caracteres se encuentre entre 3 y 40 y que sea una cadena con formato de correo electrónico.

- address: dirección de la empresa.
  - type: String

- phone: teléfono de la empresa.
  - type: String
  - required: true
  - validate: phoneValidator

Campo obligatorio pero que no tiene por qué ser único. Pasa, también, por un proceso de validación que comprueba que la cadena de caracteres cumple los estándares de los formatos de números de teléfono.

Figura 6. Ejemplo de tenant guardado en la colección tenants

2.4.3.2 course.js

Este archivo contiene el esquema para crear la colección donde se guardarán los diferentes campos de golf de la aplicación. Los documentos que se guardarán en la colección tienen los siguientes campos:

- courseid: identificador único para cada campo.
  - type: String
  - required: true
  - unique: true
  - validate: courseValidator

Este campo siempre tiene que estar presente a la hora de guardar un nuevo campo en la colección y su valor tiene que ser único. No se pueden guardar dos o más campos con el mismo identificador. Además, este campo tiene que pasar un proceso de validación antes de que el nuevo documento se guarde satisfactoriamente. En este caso el *backend* comprueba que el campo tenga siempre 5 caracteres y que esos caracteres estén entre 0 y 9.

- tenantid: identificador del *tenant* al que pertenece el campo.
  - type: String
  - required: true
  - validate: tenantidValidator

Campo obligatorio que asocia el campo a un *tenant* en concreto. Pasa por un proceso de validación como el del campo anterior se comprueba que tenga siempre 5 caracteres y que esos caracteres estén entre 0 y 9.

- holes: número de hoyos del campo.
  - type: String
  - required: true
  - validate: holesValidator

Campo obligatorio que debe pasar por un proceso de validación antes de poder guardar el documento. Este proceso comprueba que el número sea un entero de dos cifras.

- email: correo electrónico del campo.
  - type: String
  - lowercase: true
  - required: true
  - validate: emailValidator

Este campo es obligatorio y, además, tienen que estar escrito en minúsculas. Pasa por un proceso de validación que comprueba que la longitud de la cadena de caracteres se encuentre entre 3 y 40 y que sea una cadena con formato de correo electrónico.

- address: dirección del campo.
  - type: String

- phone: teléfono del campo.
  - type: String
  - validate: phoneValidator

Este campo no es obligatorio, pero pasa por un proceso de validación que comprueba que la cadena de caracteres cumple los estándares de los formatos de números de teléfono, por lo que no se puede dejar en blanco.

- longitude: longitud para ubicar el campo.
  - type: String
  - required: true
  - validate: longlatValidator

- latitude: latitud para ubicar el campo.
  - type: String
  - required: true
  - validate: longlatValidator

Estos dos campos son obligatorios pero que no tienen por qué ser únicos. Pasan por un proceso de validación que comprueba que la cadena de caracteres cumple el formato correcto para los valores de longitud y latitud.

Figura 7. Ejemplo de campo guardado en la colección courses

2.4.3.3 hole.js

Este archivo contiene el esquema para crear la colección donde se guardarán los diferentes hoyos de los campos de golf. Los documentos que se guardarán en la colección tienen los siguientes campos:

- holeid: identificador único para cada hoyo.
  - type: String
  - required: true
  - unique: true
  - validate: holeidValidator

Este campo siempre tiene que estar presente a la hora de guardar un nuevo hoyo en la colección y su valor tiene que ser único. No se pueden guardar dos o más hoyos con el mismo identificador. Además, este campo tiene que pasar un proceso de validación antes de que el nuevo documento se guarde satisfactoriamente. En este caso el *backend* comprueba que el campo tenga siempre 5 caracteres y que esos caracteres estén entre 0 y 9.

- courseid: identificador del campo al que pertenece el hoyo.
  - type: String
  - required: true
  - validate: courseidValidator

Campo obligatorio que asocia el hoyo a un campo en concreto. Pasa por un proceso de validación como el del campo anterior donde se comprueba que tenga siempre 5 caracteres y que esos caracteres estén entre 0 y 9.

- holenr: número del hoyo.
  - type: String
  - required: true
  - validate: holesValidator

- par: número de golpes para completar el hoyo.
  - type: String
  - required: true
  - validate: parValidator

Estos campos son obligatorios y deben pasar por un proceso de validación antes de poder guardar el documento. Este proceso comprueba que el número sea un entero de dos cifras.

- length: distancia desde la salida hasta el hoyo.
  - type: String
  - required: true
  - validate: lengthValidator

Similar al campo anterior, pero en este caso se comprueba que el número sea un entero o un real.

- longitude: longitud para ubicar el hoyo.
  - type: String
  - required: true
  - validate: longlatValidator

- latitude: latitud para ubicar el hoyo.
  - type: String
  - required: true
  - validate: longlatValidator

Estos dos campos son obligatorios pero que no tienen por qué ser únicos. Pasan por un proceso de validación que comprueba que la cadena de caracteres cumple el formato correcto para los valores de longitud y latitud.

Figura 8. Ejemplo de hoyo guardado en la colección holes

2.4.3.4 device.js

Este archivo contiene el esquema para crear la colección donde se guardarán los diferentes dispositivos de los campos de golf. Los documentos que se guardarán en la colección tienen los siguientes campos:

- deviceid: identificador único para cada dispositivo.
  - type: String
  - required: true
  - unique: true
  - validate: deviceidValidator

Este campo siempre tiene que estar presente a la hora de guardar un nuevo dispositivo en la colección y su valor tiene que ser único. No se pueden guardar dos o más dispositivos con el mismo identificador. Además, este campo tiene que pasar un proceso de validación antes de que el nuevo documento se guarde satisfactoriamente. En este caso el *backend* comprueba que el campo tenga siempre 5 caracteres y que esos caracteres estén entre 0 y 9.

- courseid: identificador del campo done está ubicado el dispositivo.
  - type: String
  - required: true
  - validate: courseidValidator

Campo obligatorio que asocia el dispositivo a un campo en concreto. Pasa por un proceso de validación como el del campo anterior donde se comprueba que tenga siempre 5 caracteres y que esos caracteres estén entre 0 y 9.

- type: tipo de dispositivo.
  - type: String
  - required: true

Campo obligatorio, pero no tiene que pasar un proceso de validación ya que su valor viene dado por una lista cerrada.

- status: estado del dispositivo.
  - type: String

- info: última alarma del dispositivo.
  - type: String

- longitude: longitud para ubicar el hoyo.
  - type: String

- latitude: latitud para ubicar el hoyo.
  - type: String

Estos no son obligatorios ya que será el propio dispositivo quien los rellene en la base de datos.

2.4.3.5 deviceInfo.js

Este archivo contiene el esquema para crear la colección donde se guardarán los mensajes de los dispositivos de los campos de golf. Los documentos que se guardarán en la colección tienen los siguientes campos:

- deviceid: identificador que asocia un mensaje a un dispositivo.
  - type: String
- devicetype: tipo de dispositivo asociado.
  - type: String
- message: texto de la alarma o mensaje.
  - type: String
- date: fecha de la alarma o mensaje.
  - type: String

Los campos de esta colección los rellena el dispositivo en la base de datos, pero se ha creado su esquema para crear las instancias que necesite la aplicación de esta colección.

2.4.3.6 tracker.js

Este archivo contiene el esquema para crear la colección donde se guardarán los diferentes dispositivos de rastreo que llevarán los jugadores. Los documentos que se guardarán en la colección tienen los siguientes campos:

- trackerid: identificador único para cada rastreador.
  - type: String
- playerid: nombre de usuario del jugador que lleva el rastreador.
  - type: String
- courseid: identificador del campo.
  - type: String
- inuse: indica si el dipositivo se está usando.
  - type: String
- longitude: longitud para ubicar el rastreador.
  - type: String
- latitude: latitud para ubicar el rastreador.
  - type: String

Los rastreadores no se gestionan desde la aplicación, pero se crea el esquema ya que la aplicación hace uso de las instancias de la colección.

2.4.3.7 game.js

Este archivo contiene el esquema para crear la colección donde se guardarán los diferentes partidos de los jugadores. Los documentos que se guardarán en la colección tienen los siguientes campos:

- gameid: identificador único para cada partido.
  - type: String
  - required: true
  - unique: true

Este campo siempre tiene que estar presente a la hora de guardar un nuevo partido en la colección y su valor tiene que ser único. No se pueden guardar dos o más partidos con el mismo identificador.

- courseid: identificador del campo donde se ha disputado el partido.
  - type: String
  - required: true

- username: nombre de usuario del jugador.
  - type: String
  - required: true

- holenr: número de hoyos que se han jugado en el partido.
  - type: String
  - required: true

- coursepar: par del campo.
  - type: String
  - required: true

- score: resultado del jugador.
  - type: String
  - required: true

- date: fecha en la que se ha disputado el partido.
  - type: String
  - required: true

La introducción de los datos de los partidos no está implementada en la actual versión de la aplicación. En el escenario que se ha creado esta aplicación esta información se introduce en la base de datos desde otros dispositivos, en cualquier caso, se ha creado un posible esquema por si en algún momento los datos de los partidos se pudieran introducir desde la aplicación.

2.4.3.8 gamehole.js

Este archivo contiene el esquema para crear la colección donde se guardarán los detalles de los partidos de los jugadores. Los documentos que se guardarán en la colección tienen los siguientes campos:

- gameholeid: identificador del hoyo.
  - type: String
  - required: true
  - unique: true

Este campo siempre tiene que estar presente a la hora de guardar un nuevo hoyo en la colección y su valor tiene que ser único. No se pueden guardar dos o más hoyos con el mismo identificador. Este identificador es el mismo que el identificador de un hoyo de la colección *holes*.

- gameid: identificador partido donde se ha jugado el hoyo.
  - type: String
  - required: true

- hole: número del hoyo.
  - type: String
  - required: true

- holepar: par del hoyo.
  - type: String
  - required: true

- score: resultado del jugador.
  - type: String
  - required: true

Al igual que en el caso anterior la introducción de los datos de esta colección no está implementada en la actual versión de la aplicación. En el escenario que se ha creado esta aplicación esta información se introduce en la base de datos desde otros dispositivos, en cualquier caso, se ha creado un posible esquema por si en algún momento los datos de los partidos se pudieran introducir desde la aplicación.

2.4.3.9 user.js

Este archivo contiene el esquema para crear la colección donde se guardarán los usuarios de la aplicación. Los documentos que se guardarán en la colección tienen los siguientes campos:

- username: identificador único para cada usuario.
  - type: String
  - required: true
  - unique: true
  - lowercase: true
  - validate: usernameValidator

Este campo siempre tiene que estar presente a la hora de guardar un nuevo usuario en la colección y su valor tiene que ser único. No se pueden guardar dos o más usuarios con el mismo nombre de usuario. Además, este campo tiene que pasar un proceso de validación antes de que el nuevo documento se guarde satisfactoriamente. En este caso el *backend* comprueba que el campo tenga siempre tenga entre 3 y 25 caracteres y que sea una cadena alfanumérica.

- password: contraseña del usuario.
  - type: String
  - required: true
  - select: false
  - validate: passwordValidator

Campo obligatorio y que no por defecto no puede ser accedido a través de las consultas a la base de datos. Además, pasa por un proceso de validación que comprueba que el campo tenga entre 8 y 35 caracteres y que la cadena contenga al menos una mayúscula, una minúscula, un número y un carácter especial.

- name: nombre real del usuario.
  - type: String
  - required: true
  - validate: nameValidator

Campo obligatorio que no tiene por qué ser único. Pasa un proceso de validación que comprueba que el campo tenga entre 3 y 30 caracteres y que la cadena no contenga números ni caracteres especiales.

- email: correo electrónico del usuario.
  - type: String
  - lowercase: true
  - required: true
  - unique: true
  - validate: emailValidator

Este campo tiene unas características idénticas al campo *username*. Es obligatorio, debe de ser único, y tiene que estar escrito en minúsculas. Pas por un proceso de validación que comprueba que la longitud de la cadena de caracteres se encuentre entre 3 y 40 y que sea una cadena con formato de correo electrónico.

- tenant: identificador del *tenant* al que pertenece el usuario.
  - type: String
  - required: true
  - default: '99999'
  - validate: nameValidator

Campo obligatorio que no tiene por qué ser único y que por defecto tiene el valor 99999. Pasa un proceso de validación que comprueba que el campo tenga siempre 5 caracteres y que estos estén entre el 0 y el 9.

- address: dirección del usuario.
  - type: String

- phone: teléfono del usuario.
  - type: String

- active: indica si la cuenta de usuario está activada.
  - type: Boolean
  - required: true
  - default: false

Campo obligatorio que por defecto tiene el valor *false*. No pasa por ningún tipo de proceso de validación.

- temporarytoken: token temporal que se utiliza para activar la cuenta.
  - type: String
  - required: true

Campo obligatorio que no pasa por ningún tipo de proceso de validación.

- resettoken: token temporal que se utiliza para reiniciar la contraseña.
  - type: String
  - required: true

Campo obligatorio que no pasa por ningún tipo de proceso de validación.

- permission: nivel de privilegios del usuario
  - type: String
  - required: true
  - default: 'user'

Campo obligatorio que por defecto tiene el valor 'user'. No pasa por ningún tipo de proceso de validación.

3. Tecnologías y lenguajes

3.1 Tecnologías utilizadas

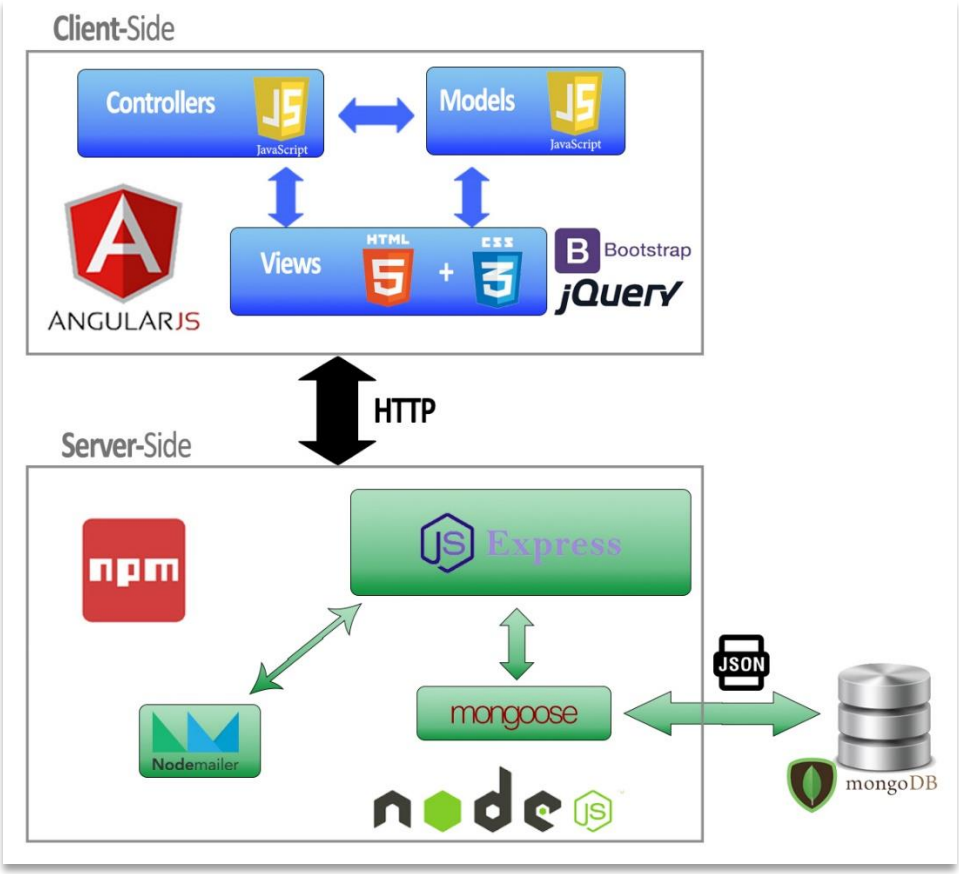


Figura 9. Tecnologías y lenguajes

3.2 Node

Node.js[1] es un entorno en tiempo de ejecución JavaScript multiplataforma orientado a eventos asíncronos y está diseñado para la creación de aplicaciones en red escalables. Se basa en el motor V8 de Google lo que lo hace muy liviano y eficiente.

La velocidad y gestión eficiente de los datos de la base de datos es un punto crítico en el desarrollo de la aplicación, y es en este punto donde destaca Node, ya que permite ejecutar el código de forma extremadamente rápida.

Algunas de las características más destacadas de Node son:

- Diseñado para optimizar el rendimiento y la escalabilidad de las aplicaciones web. Lo que lo convierte en una gran opción para abordar muchos de los problemas del desarrollo web, como por ejemplo las aplicaciones web en tiempo real.

- Escrito en “Simple JavaScript”, lo que se traduce en menos tiempo perdido tratando con los “cambios de contexto” entre lenguajes del lado del servidor y del cliente.

- JavaScript[2] es un lenguaje de programación relativamente nuevo, lo que le permite beneficiarse de los avances en el diseño de lenguajes comparado con otros lenguajes como Python, PHP y otros. Existen conversiones o compilaciones de muchos lenguajes nuevos y populares a JavaScript de manera que también se pueden usar en Node.

- El gestor de paquetes de Node (NPM del inglés: *Node Packet Manager*) proporciona acceso a gran cantidad de paquetes reutilizables en diferentes proyectos.

- Es portable, con versiones para Microsoft Windows, OS X, Linux, Solaris, FreeBSD, OpenBSD, WebOS, y NonStop OS.

- La comunidad de desarrolladores es muy activa, prestando ayuda a todo aquel que la necesite.

3.2.1 NPM

Node utiliza el ecosistema de paquetes NPM, el ecosistema más grande de librerías de código abierto en el mundo.

NPM (*Node Package Manager*, administrador de paquetes de Node) es un administrador de paquetes, un software que automatiza el proceso de instalación, actualización, eliminación y configuración de componentes en un sistema.

Un paquete (*package*) es el término formal para referirse a un componente, ya sea de software, código fuente o documentación, el cual se puede instalar en nuestro sistema. Los paquetes facilitan la organización de la aplicación, así como su reutilización. En Node, un paquete se implementa mediante un directorio, el cual contiene el código fuente del dicho componente.

En la actualidad existen diferentes administradores de paquetes como, por ejemplo, *APT, RPM o NPM*. Pero para la aplicación que se va a desarrollar en este TFG se utilizará NPM, ya que facilita el uso y la instalación de paquetes en Node.

NPM se instala automáticamente cuando se instala Node, y viene con herramientas o comandos con los que realizar operaciones de instalación, actualización, desinstalación, consulta, etc. Siendo el comando *npm* uno de los más importante a la hora de utilizar este gestor de paquetes.

El comando *npm* es una aplicación cliente y una utilidad de línea de comandos que nos permite, entre otras cosas, instalar paquetes en nuestra máquina. Se encuentra escrita íntegramente en Node y se publica en el propio repositorio NPM mediante el paquete del mismo nombre. Sus principales funciones son:

- Instalar, actualizar y desinstalar paquetes.
- Consultar el índice de paquetes.
- Publicar paquetes en el repositorio.

Para consultar todos los paquetes que se pueden instalar desde NPM solo hay que realizar una búsqueda en su sitio web npmjs.org.

Para poder utilizar un paquete de NPM en un determinado proyecto, es necesario instalarlo en el disco de nuestra máquina. Para ello, se solicita una copia del módulo al repositorio NPM y se instala. Ambas cosas se hacen mediante el comando *npm install*.

Los paquetes se pueden instalar de forma global o local, siendo la instalación global la que permite que varios proyectos hagan uso de dicho paquete, mientras que con la instalación local el paquete se instala en una ubicación únicamente accesible por un proyecto.

3.2.2 Package.json

Package.json es el archivo de configuración de un proyecto de Node. En este archivo, que debe estar en la raíz del proyecto, va a quedar reflejada la configuración del proyecto. Este archivo puede crearse de forma manual o mediante el asistente de npm.

Este fichero puede ser un documento en formato JSON, o un objeto literal de JavaScript, y su objetivo es documentar los paquetes Node que se utilizarán en la aplicación. Este fichero permite llevar un control de los paquetes instalados y sus versiones, pero, además, es necesario si se desea publicar una aplicación en el listado de paquetes de npm. A continuación, Se desgranan las propiedades más importantes que forman este fichero.

- *Name*: esta propiedad es obligatoria, indica el nombre de la aplicación y es preferible que esté en minúscula.

- *Version*: indica la versión de la aplicación. su valor es un número con el formato MAJOR.MINOR.PATCH. Siendo cada uno:
  - MAJOR: cambios de gran envergadura, donde versiones anteriores se hacen incompatibles con la nueva.
  - MINOR: cambios de menor envergadura en los que se añaden funcionalidad que no provoca incompatibilidades entre versiones.
  - PATCH: cambios menores, o parches, en los que se solucionan fallos (bugs). Tampoco provoca incompatibilidades.

- *Description*: breve y concisa descripción de la aplicación. Normalmente se utiliza para explicar rápidamente la funcionalidad principal del proyecto.

- *Main*: indica el punto de entrada principal de la aplicación. En general se recomienda tener un único script principal.

- *Author*: nombre del autor de la aplicación.

- *Scripts*: utilizado para agregar código que pueda ser ejecutado como un comando más en el terminal, vía npm. Compuesto por la clave, que indica el nombre del comando agregado, y el valor, que contiene el código que será ejecutado cuando se utilice dicho comando. El script más habitual es *start*, el cual indica el comando inicial de la aplicación.

- *Keywords*: palabras clave para poder facilitar la búsqueda de la aplicación en el listado de paquetes de npm.

- *Dependencies*: listado de las dependencias del proyecto. Es decir, otros módulos que son necesarios para el funcionamiento de nuestra aplicación y que serán instalados automáticamente cuando los usuarios instalen el paquete/aplicación.

- devDependencies: muestra las dependencias que son utilizadas específicamente para labores de desarrollo y testeo de la aplicación.

- *Private*: Contiene un valor booleano, que es utilizado para indicar la visibilidad de nuestro paquete desde el listado de paquetes *npm*. Si el valor es verdadero (*true*) el paquete no se publicará.

- *Engines*: información sobre la versión de Node.js para la que ha sido desarrollada la aplicacón, y sobre la cual se asegura que no habrá incompatibilidades. Si no especifica, npm asumirá que la aplicación funciona en Node.

- *License*: indica bajo qué licencia de código abierto se encuentra el proyecto.

```
{
  "name": "tgolf_app",
  "version": "1.0.0",
  "description": "tgolf application",
  "main": "server.js",
  "scripts": {
    "start": "node server.js"
  },
  "keywords": [
    "MEAN",
    "stack",
    "golf"
  ],
  "author": "Gorka Estevez",
  "license": "ISC",
  "dependencies": {
    "bcrypt-nodejs": "0.0.3",
    "body-parser": "^1.19.0",
    "express": "^4.17.1",
    "express-session": "^1.17.0",
    "jsonwebtoken": "^8.5.1",
    "mongoose": "^5.7.13",
    "mongoose-title-case": "^0.1.1",
    "mongoose-validator": "^2.1.0",
    "morgan": "^1.9.1",
    "nodemailer": "^6.3.1",
    "nodemailer-sendgrid-transport": "^0.2.0",
    "socket.io": "^2.3.0"
  },
  "devDependencies": {
    "bootstrap4-card-tables": "^1.2.1"
  }
}
```

Figura 10. Archivo package.json

3.2.3 Express

Hoy en día Express[1] es el *framework* web más popular de Node. Es, además, la librería subyacente de multitud de *frameworks* web que se utilizan en Node. Express proporciona mecanismos:

- Controlador de escritura para peticiones HTTP con diferentes llamadas en diferentes rutas URL.

- Integración con diferentes motores de renderización de "vistas", de manera que se puedan generar respuestas mediante la introducción de datos en plantillas o modelos.

- Establecimiento de los ajustes más comunes de las aplicaciones web, como el puerto de conexión con la aplicación, y la localización de las plantillas o modelos que se utilizarán para renderizar las respuestas.

- Inserción de middleware de procesamiento de peticiones adicional en cualquier punto dentro del pipeline del controlador de peticiones.

En las webs tradicionales asociadas a bases datos, la aplicación web espera a las peticiones HTTP por parte del cliente y en función del patrón de la URL y alguna información que se pueda extraer de las llamadas de la petición, decide que acción se debe llevar a cabo. Tras completar la acción requerida la aplicación devolverá la respuesta al cliente, habitualmente un navegador web, creando de forma dinámica una página HTML con el resultado de la petición.

Express provee métodos para especificar que función se debe utilizar para cada llamada HTTP (GET, POST, PUT, etc.) y patrón de URL, así como métodos que especifican qué motor de vistas se debe usar, dónde están almacenadas las vistas y qué vista mostrar con la respuesta.

Aunque *Express* es en sí mismo bastante minimalista, la comunidad de desarrolladores ha creado paquetes de middleware compatibles que permiten solucionar casi cualquier problema de en el ámbito del desarrollo web. Por poner algunos ejemplos, existen librerías para trabajar con cookies, sesiones, inicios de sesión de usuario, parámetros URL, etc.

3.3 MongoDB

MongoDB[3] es una base de datos distribuida NoSQL de tipo documental que almacena los datos en documentos de tipo JSON. Está escrita en C++, es multiplataforma, de código abierto y gratuita.

El proyecto nació a finales del año 2007 como un proyecto interno de una empresa llamada 10Gen para usarlo en como parte del desarrollo de una plataforma como servicio, pero en 2009 decidieron liberarlo bajo la licencia de código abierto AGPL y dedicarse íntegramente a él, ofreciendo soporte comercial y servicios relacionados. Su nombre proviene de la palabra inglesa *humongous*, que significa "enorme", y hace referencia a la capacidad de MongoDB de gestionar cantidades enormes de datos.

Sus principales características[4] son:

- Proporciona gran rendimiento en la persistencia de datos. El soporte para modelos de datos embebidos reduce la actividad de las entradas y salidas en la base de datos. Y los índices soportan consultas más rápidas.

- Soporta un lenguaje de consultas enriquecido para ofrecer operaciones de lectura y escritura como: agregación de datos, búsquedas de texto o consultas geoespaciales.

- Proporciona alta disponibilidad mediante el uso de grupos de servidores MongoDB (*replica set*) los cuales garantizan la redundancia y la accesibilidad de los datos.

- Ofrece escalabilidad horizontal como parte de su funcionamiento básico. El método de *sharding* distribuye los datos dentro de un clúster de máquinas y, además, se permite el uso de zonas de datos.

Las bases de datos documentales como MongoDB[5] se pueden utilizar en muchos tipos de escenarios diferentes, pero son especialmente útiles cuando se necesita flexibilidad en la definición de los datos y facilidad de acceso a estos, gran rendimiento o hacer crecer la aplicación rápidamente.

Este tipo de bases de datos son muy adecuadas para crear aplicaciones web que registren muchos datos o que se quieran crear de manera flexible, pero también son válidas para sistemas grandes como aplicaciones de registro de datos de sensores, sistemas gestores de datos de ventas, aplicaciones de análisis de datos, almacenamiento para redes sociales, etc.

Al estar basada en JavaScript, es una buena opción para el desarrollo de aplicaciones web del tipo Single-Page Application. En general, MongoDB puede utilizarse en prácticamente cualquier situación en la que se usarían bases de datos relacionales como SQL Server o MySQL, pero sin la rigidez que presentan estas.

En MongoDB, cada registro o conjunto de datos se conoce como documento, y estos pueden ser agrupados en colecciones. Los documentos siempre tendrán un campo llamado "_id", cuyo valor genera automáticamente MongoDB, y que sirve para diferenciarlos. Las colecciones podrían ser el equivalente a las tablas en las bases de datos relacionales, pero no tienen un esquema fijo y pueden almacenar datos con diferentes formatos. Aunque en la práctica no conoceremos el formato de los datos ya que trabajaremos sobre documento de tipo JSON tanto para guardar como consultar esos datos.

La principal diferencia jerárquica con los SGBDR, (Sistemas Gestores de Bases de Datos Relacionales), es que no se utilizan tablas, filas, ni columnas, se utilizan documentos con distintas estructuras. Un conjunto de "Campos" formarán un "Documento", que en caso de asociarse con otros formarán una "Colección". Por lo tanto, en un escenario donde se esté utilizando MongoDB, las bases de datos estarán formadas por "Colecciones", y a su vez, cada servidor podrá contener tantas bases de datos como el equipo lo permita.

A primera vista podría parecer que trabajando con bases de datos NoSQL se llegaría al punto de tener una gran cantidad de datos difíciles de relacionar y, por ende, difíciles de gestionar. Pero en realidad, se trabaja con documentos que siguen una estructura que se fija al principio del proyecto, teniendo la posibilidad de alterar esta estructura de forma flexible para los documentos que se desee si esto favorece el desarrollo o el uso de la aplicación.

Figura 11. Documento de MongoDB

3.4 Angular

AngularJS[6] es un proyecto de código abierto, realizado en JavaScript mediante el cual se proporcionan un conjunto de librerías que facilitan el desarrollo de aplicaciones web, proponiendo, además, un modelo o patrón de diseño. Se puede decir que AngularJS[7] es un *framework* para el desarrollo, más concretamente, un *framework* de desarrollo para JavaScript con programación del lado del cliente.

AngularJS se basa en el patrón MVC (Modelo Vista Controlador)[8] que sirve para definir la organización y estructura de la aplicación.

- Modelo: representa la estructura de los datos de la aplicación. No se debe confundir con el modelo de la base de datos de la aplicación.

- Vista: la parte del código que se encargará de mostrar al usuario la aplicación y le permitirá interactuar con ella. Las vistas serán las páginas en código HTML[9] de la aplicación.

- Controlador: es el intermediario entre las vistas y el modelo. Los controladores serán los encargados de gestionar toda la lógica de la aplicación.

En Angular tiene una gran importancia un objeto JavaScript denominado *$scope* que representa el modelo. Este objeto será el encargado de comunicar la parte HTML con la parte JavaScript y viceversa. De forma muy simplificada se podría decir que Angular se basará en los cambios que ocurran en el *$scope* para actualizar la vista, y a la inversa, actualizará el $*scope* con los cambios que se sucedan en la vista.

3.4.1 jQuery

jQuery[10] es una biblioteca de JavaScript que permite simplificar la manera de interactuar con documentos HTML, manejar eventos, manipular el árbol DOM (*Document Object Model*), desarrollar animaciones y agregar interacción con la técnica AJAX. jQuery es software libre y de código abierto y, como otras bibliotecas de JavaScript, permite grandes resultados en menos tiempo y espacio.

El éxito de jQuery[11] reside en sus capacidades multiplataforma, corrigiendo de manera automática los errores y ejecutándose de la misma forma en los navegadores más populares.

Además, facilita el uso de AJAX, funcionando de manera asíncrona al resto del código, lo que permite al código AJAX comunicarse con el servidor y actualizar los datos sin necesidad de recargar la página.

Esta biblioteca también permite manipular el DOM de una página HTML, permitiendo, entre otras cosas, insertar o eliminar elementos, o agrupar líneas de forma sencilla.

Las animaciones también se crean de manera más fácil gracias a jQuery, ya que permite crearlas con unas pocas líneas de código donde lo más importante es insertar las variables.

En definitiva, a la hora de programar en JavaScript se hace casi obligatorio adquirir conocimientos de jQuery ya que facilita mucho la labor del desarrollador.

3.4.2 Bootstrap

Bootstrap[12] es un *framework* CSS y Javascript que nace con la intención de permitir la creación de sitios o aplicaciones web con interfaces limpias y diseños adaptables. Para conseguir su cometido, ofrece una gran variedad de herramientas y funciones para que los desarrolladores puedan crear cualquier tipo de sitios o aplicaciones.

El éxito de Bootstrap se debe a que permite crear sitios o aplicaciones web totalmente adaptables a cualquier dispositivo. Característica sumamente importante hoy en día, teniendo en cuenta que los usuarios acceden a internet desde una gran variedad de dispositivos.

Las características principales[13] de este *framework*:

- Creación de interfaces 100% adaptables. Las aplicaciones creadas con Bootstrap se adaptan a los diferentes navegadores y resoluciones tanto de equipos de sobremesa como de dispositivos móviles.
- Integración perfecta con las principales librerías de JavaScript.
- Totalmente compatible con los estándares CSS3 y HTML5.
- Es ligero, y permite una integración limpia con el proyecto que se esté desarrollando.
- Funciona en los navegadores más populares, incluso en algunas versiones antiguas o no soportadas oficialmente.
- Existen gran cantidad de plantillas temáticas que se pueden adquirir para facilitar el trabajo de diseño de la aplicación.

3.4.3 Leaflet

Leaflet[14] es una librería JavaScript de código abierto diseñada para crear mapas interactivos tanto en aplicaciones web de escritorio como en aquellas que se deban usar en dispositivos móviles. Pese a su poco peso (unos 38KB) ofrece todas las características para satisfacer cualquier necesidad de los desarrolladores de aplicaciones web.

Esta librería está desarrollada teniendo en cuenta la simplicidad, el rendimiento y la usabilidad, por lo que funciona de forma eficiente en la mayoría de las plataformas de escritorio y móviles. Dada su filosofía de código abierto, las funcionalidades de Leaflet pueden aumentarse con gran cantidad de plugins y permite un gran nivel de personalización.

3.4.4 Datatables

Datatables[15] es un plugin de código abierto para la librería Javascript de jQuery altamente personalizable que permite renderizar tablas HTML con funcionalidades avanzadas como:

- Paginación: navegación de páginas con funciones de “previo” y “siguiente”.
- Búsqueda instantánea: mientras se escribe en el cuadro de búsqueda la tabla va actualizándose para ofrecer el resultado de la búsqueda.
- Ordenamiento múltiple: Datatables permite ordenar los datos por varias columnas al mismo tiempo.
- Admite múltiples fuentes de datos.
- Fácilmente personalizable.
- Gran variedad de extensiones.
- Las tablas renderizadas se pueden adaptar a todo tipo de dispositivos, desde equipos de escritorio como dispositivos móviles.
- Facilidad para ofrecer la tabla en diferentes idiomas.

4. Implementación

4.1 api.js

La carpeta *routes* se crea para definir las rutas donde se responda las solicitudes del cliente. En este directorio se encuentra el archivo api.js en el cual se definen todas las rutas.

El archivo está organizado de la siguiente manera:

- En primer lugar, las rutas para el registro de empresas, campos, hoyos y usuarios con sus correspondientes rutas de comprobación de comprobación de identificadores, correos, o nombres de usuario.

- En segundo lugar, se encuentran las rutas relacionadas con el login de los usuarios. La ruta para autenticar al usuario, para activar la cuenta, para reiniciar la contraseña o recuperar el nombre de usuario entre otras.

- Y, en tercer lugar, se definen todas las rutas relacionadas con la gestión de usuarios, empresas, campos, etc. Estas rutas son las encargadas de obtener de mongoDB los datos necesarios para llenar tablas, eliminar documentos de la base de datos o editar documentos ya existentes.

Para que todas las rutas funcionen correctamente es necesario definir las variables de los esquemas de la base de datos para poder realizar las consultas mediante los métodos que ofrece la librería *mongoose*[16].

```
//TENANT MANAGEMENT
router.get('/tenantmgmt', function(req, res) {
  User.findOne({ username: req.decoded.username }, function(err, mainUser) {
    if (err) throw err;
    if (!mainUser) {
      res.json({ success: false, message: 'No user found.' });
    } else {
      if (mainUser.permission === 'admin') {
        Tenant.find({}, function(err, tenants) {
          if (err) throw err;
          if (!tenants.length) {
            res.json({ success: false, message: 'No tenant found.'});
          } else {
            res.json({ success: true, tenants: tenants, permission: mainUser.permission });
          }
        });
      } else if (mainUser.permission === 'admintenant') {
        Tenant.findOne({ tenantid: mainUser.tenant }, function(err, tenants) {
          if (err) throw err;
          if (!tenants) {
            res.json({ success: false, message: 'No tenant found.'});
          } else {
            res.json({ success: true, tenants: tenants, permission: mainUser.permission});
          }
        });
      } else {
        res.json({ success: false, message: 'Insufficient permission level.'});
      }
    }
  });
});
```

Figura 12. Parte del código del archivo api.js

4.2 Controladores

En la carpeta *controllers* se encuentran todos los controladores necesarios para unir las vistas con el modelo. Se pueden tener tantos archivos de controladores como se quiera, pero para la aplicación que se ha desarrollado se ha optado por crear archivos que engloben controladores afines o relacionados, para mantener un orden dentro del código.

Los controladores que se han creado para la aplicación son los siguientes:

- courseCtrl.js: este archivo contiene controladores relacionados con los campos de golf:
  - courseCtrl: controlador encargado de registrar nuevos campos.
  - liveGameCtrl: controlador encargado de generar el mapa para el control del juego en tiempo real.
  - playerStatisticsCtrl: controlador encargado de obtener los datos para tabla de los partidos de un jugador, y de permitir el borrado de los partidos.
  - gameStatsCtrl: controlador encargado de obtener los datos para la tabla del detalle de los partidos de los jugadores.
  - currentGameCtrl: controlador encargado de obtener los datos con los que se rellenará la tabla del partido actual de un jugador.

- courseMgmtCtrl.js: este archivo contiene los controladores para la gestión de los campos:
  - courseMgmtCtrl: controlador encargado de obtener los datos para tabla de los campos, y de permitir el borrado de los estos.
  - editCourseCtrl: controlador encargado de la edición de los datos de los campos.
  - manageHolesCtrl: controlador encargado de obtener los datos para tabla de los hoyos de un campo, de registrar nuevos hoyos y de permitir el borrado de los estos.
  - editHoleCtrl: controlador encargado de la edición de los datos de los hoyos.

- deviceCtrl.js: este archivo contiene los controladores relacionados con la visualización de los datos de los dispositivos.
  - deviceCtrl: controlador encargado de obtener los datos para las tablas de estado de los dispositivos y del log de alarmas.
  - deviceMapCtrl: controlador encargado de generar el mapa con la situación y el estado de los dispositivos.

- deviceMgmtCtrl.js: este archivo contiene los controladores para la gestión de los dispositivos instalados en los campos.
  - regDeviceCtrl: controlador encargado de registrar un nuevo dispositivo.
  - deviceMgmtCtrl: controlador encargado de obtener los datos para la tabla de los dispositivos dados de alta, y de permitir el borrado de estos.

    - editDeviceCtrl: controlador encargado de la dedición de los datos de un dispositivo.

- emailCtrl.js: este archivo contiene los controladores relacionados con la activación de la cuenta, la recuperación del nombre de usuario o el reinicio de la contraseña.
    - emailCtrl: controlador encargado de activar la cuenta de usuario.
    - resendCtrl: controlador encargado de enviar al usuario un nuevo enlace de activación de la cuenta.
    - usernameCtrl: controlador encargado de enviar al usuario su nombre de usuario.
    - passwordCtrl: controlador encargado de enviar al usuario el enlace de reinicio de la contraseña.
    - resetCtrl: controlador encargado del reinicio de la contraseña.

- mainCtrl.js: este archivo contiene un único controlador, mainCtrl, que se encarga del inicio y desconexión de la sesión del usuario, y de comprobar periódicamente, cada 15 minutos, si la sesión del usuario ha expirado.

- tenantCtrl.js: este archivo contiene un único controlador, tenantCtrl, que es el encargado de registrar los nuevos *tenant* en el sistema.

- tenantMgmtCtrl.js: este archivo contiene los controladores relacionados con la gestión de los *tenant*.
    - tenantMgmtCtrl: controlador encargado de obtener los datos para tabla de los *tenant*, y de permitir el borrado de estos.
    - editTenantCtrl: controlador encargado de la edición de los datos de un *tenant*.

- userCtrl.js: este archivo contiene un único controlador, regCtrl, que se encarga de registrar los nuevos usuarios en el sistema. Además, contiene una directiva que se encarga de verificar si los dos campos de contraseña coinciden.

- usersMgmtCtrl.js: archivo que contiene los controladores relacionados con la gestión de los usuarios.
    - usersMgmtCtrl: contorlador encargado de obtener los datos para la tabla de usuarios, y de permitir el borrado de estos.
    - editCtrl: controlador encargado de la edición de los datos de un usuario.

4.3 Servicios

En la carpeta *services* se encuentran todos los archivos que contienen la colección de funciones que se utilizan en los controladores y la ruta que se utilizará en el archivo *api.js* para obtener la respuesta.

Estos archivos están divididos en función de a qué dan servicio, es decir, habrá un archivo para las funciones de usuarios, otro para las funciones de los campos, etc.

Estas funciones se crean dentro de unas "factorías" (*factory* en inglés) donde se les dará un nombre y la ruta dentro del archivo *api.js* a donde tienen que apuntar y donde se ejecutará la función.

```
1   angular.module('userServices', [])
2
3   .factory('User', function($http) {
4     var userFactory = {};
5
6     //User.create(regData)
7     userFactory.create = function(regData, creation) {
8       if (creation === 'creator') {
9         regData.creation = creation;
10        return $http.post('/api/users', regData);
11      }
12      if (creation === 'register') {
13        regData.creation = creation;
14        return $http.post('/api/users', regData);
15      }
16    };
17
18    //User.checkUsername(regData);
19    userFactory.checkUsername = function(regData) {
20      return $http.post('/api/checkusername', regData);
21    };
22
23    //User.checkEmail(regData);
24    userFactory.checkEmail = function(regData) {
25      return $http.post('/api/checkemail', regData);
26    };
27
28    //User.activateAccount(token);
29    userFactory.activateAccount = function(token) {
30      return $http.put('/api/activate/' + token);
```

Figura 13. Parte del código del archivo userServices.js

4.4 Vistas

En la carpeta *views* se encuentran todas las vistas o páginas de las que se compone la aplicación. Estas vistas son archivos HTML y son las encargadas de mostrar al usuario la aplicación en la pantalla de su navegador.

Las vistas se han ordenado por grupos: todas las relacionadas con los campos, incluidas las de los hoyos, bajo la carpeta *courses*; aquellas relacionadas con los dispositivos bajo la carpeta *maintenance*; las relacionadas con los *tenant* bajo la carpeta *tenants*; y todas las vistas relacionadas con los usuarios bajo la carpeta *users*.

A pesar de este ordenamiento, existen dos vistas que no pertenecen a ningún grupo: *start.html* e *index.html*. La primera de ellas es la página de inicio de la aplicación. Si no hay ningún usuario autenticado en la aplicación al acceder a la dirección de esta, aparecerá siempre esta página, en la que se deberán introducir las credenciales para poder acceder. La segunda vista, *index.html* es una vista especial ya que será la encargada de referenciar todos los scripts y estilos para que la aplicación cargue correctamente. Además, cargará la "vista general", es decir, el panel de navegación, el pie de página y el contenedor donde se irán cargando el resto de las vistas.

Como ya se ha mencionado anteriormente, las vistas se han programado con HTML y se ha utilizado CSS para gestionar los estilos. Además, se ha utilizado dataTables para enriquecer las tablas de la aplicación y para facilitar la creación de estas, y se han utilizado los *modals* de Bootstrap para añadir mensajes emergentes con información relevante sobre las acciones de los usuarios o como menús emergentes.

```
<div id="deviceModal" class="modal fade" tabindex="-1">
    <div class="modal-dialog">
        <div class="modal-content">
            <div class="modal-header">
                <h5 class="modal-title">Alarms log</h5>
                <button type="button" class="close" data-dismiss="modal">&times;</button>
            </div>
            <div class="modal-body">
              <div class="text-center show-hide-message" ng-show="device.errorMsg">
                <div class="alert alert-danger">{{ device.errorMsg }}</div>
              </div>
            </div>
        </div>
    </div>
</div>

<table id="alarmsTable" class="table table-striped table-bordered table-sm" cellspacing="0" width="100%">
  <thead>
    <tr>
      <th class="th-sm">Device ID</th>
      <th class="th-sm">Device Type</th>
      <th class="th-sm">Date</th>
      <th class="th-sm">Message</th>
    </tr>
  </thead>
  <tbody>
    <tr ng-repeat="alarm in alarms">
      <td>{{ alarm.deviceid }}</td>
      <td>{{ alarm.devicetype }}</td>
      <td>{{ alarm.date }}</td>
      <td>{{ alarm.message }}</td>
```

Figura 14. Parte del código para la vista alarms.html

4.5 Otros directorios y archivos

En esta sección se comentan otros archivos o directorios de interés.

- Directorio assets: en esta carpeta se encuentran activos utilizados en la aplicación.
  - Directorio css: en esta carpeta se almacenan todos los archivos de estilos.
  - Directorio fonts: lugar donde se almacenan las fuentes de la aplicación.
  - Directorio images: aquí se almacenan las imágenes de la aplicación.
  - Directorio js: carpeta para almacenar los diferentes scripts o *plugins* que no se encuentren en la carpeta node_modules.

- routes.js: en este archivo se definen todas las rutas de las vistas. Se indica a la aplicación que vista cargar cuando se llame a una ruta en concreto y, además, se define el controlador que utilizará la vista, y si es necesario estar autenticado y con qué permiso, para poder mostrar la vista solicitada.

```
.when('/changepassword/:username/:id', {
  templateUrl: 'app/views/pages/users/management/changepassword.html',
  controller: 'resetCtrl',
  controllerAs: 'reset',
  authenticated: true
})

.when('/usercreation/:creator/:tenantid', {
  templateUrl: 'app/views/pages/users/management/usercreation.html',
  controller: 'regCtrl',
  controllerAs: 'register',
  authenticated: true,
  permission: ['admin', 'admintenant', 'caddiemaster']
})
```

Figura 15. Parte del archivo routes.js

- server.js: el archivo que inicia la aplicación y realiza la conexión con la base de datos mongoDB. En esta versión de la aplicación la conexión con la base de datos es utilizando un usuario administrador de la misma.

```
const mongoOptions = {
  useNewUrlParser: true,
  useUnifiedTopology: true,
  useFindAndModify: false
};
mongoose.set('useUnifiedTopology', true );
mongoose.connect('mongodb+srv://admin:4uqTMEfxh0HUGzGO@tgolf-46ona.mongodb.net/tgolf_db?retryWrites=true&w=majority', mongoOptions, function (err) {
  if (err) {
    console.log('Not connected to the database: ' + err);
  } else {
    console.log('Succesfully connected to MongoDB');
  }
});
```

Figura 16. Conexión con la base de datos

5. Resultado

Este apartado pretende describir el resultado o producto final obtenido, es decir, la aplicación web. Se ha optado por un modelo Single-Page Application o SPA. Una SPA es una aplicación web que se ejecuta bajo una única página. El contenido de esta página cambia dinámicamente una parte de esta permitiendo reutilizar, por ejemplo, el encabezado y pie de página o los menús de navegación. Por lo tanto, a medida que el usuario interactúa con la aplicación la página irá cargando, si hicieran falta, los recursos necesarios para cumplimentar lo solicitado por el usuario.

En el caso de la aplicación desarrollada en este trabajo, la página principal constará de un panel de navegación, un contenedor principal y pie de página o *footer* comunes a todas las vistas. El contenido del contenedor principal irá cambiando en función de la demanda del usuario y los botones o pestañas del panel de navegación cambiarán en función del usuario registrado. El pie de página estará formado por el copyright y los enlaces a las políticas de privacidad y los términos de uso de la página.

Como ejemplo, se muestra cómo sería una página cualquiera de la aplicación a desarrollar:

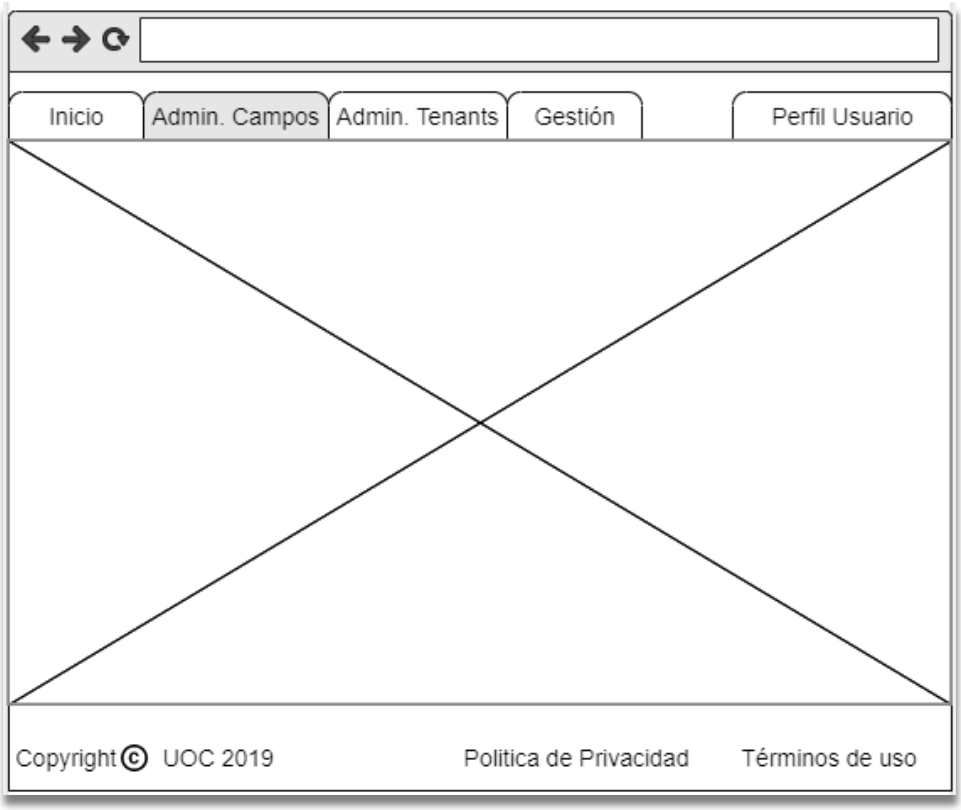


Figura 17. Boceto del diseño de una vista cualquiera

5.1 Panel de navegación

El panel de navegación de la aplicación será común para todas las vistas, pero su contenido irá cambiando en función de los privilegios o permisos que tenga el usuario. Este panel de navegación desaparecerá si a través del cliente web no se ha autenticado ningún usuario en la aplicación.

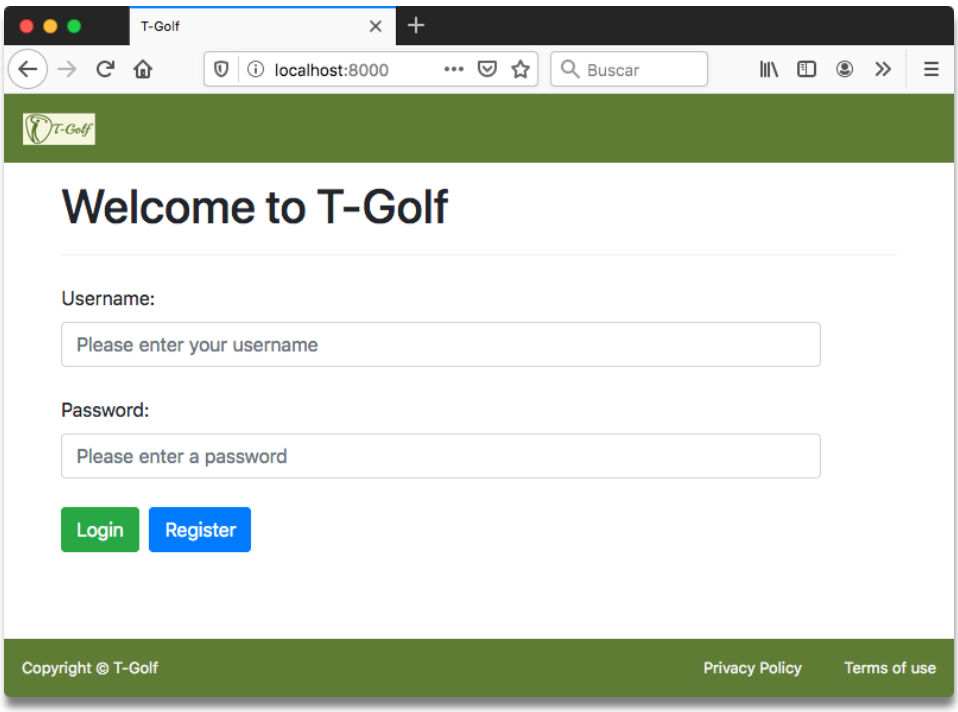


Figura 18. Pantalla de login. El panel de navegación no es visible

Cuando un usuario se autentica en la aplicación, el panel de navegación se hace visible mostrando las opciones en función del usuario autenticado. Además, la página principal muestra un resumen de las acciones que puede realizar cada usuario en función de sus permisos.

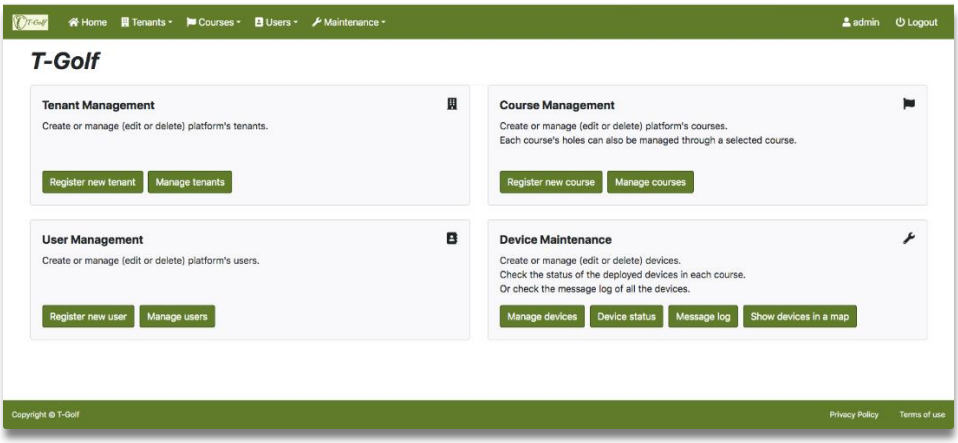


Figura 19. Pantalla de inicio del usuario administrador del sistema

5.1.1 Jugador

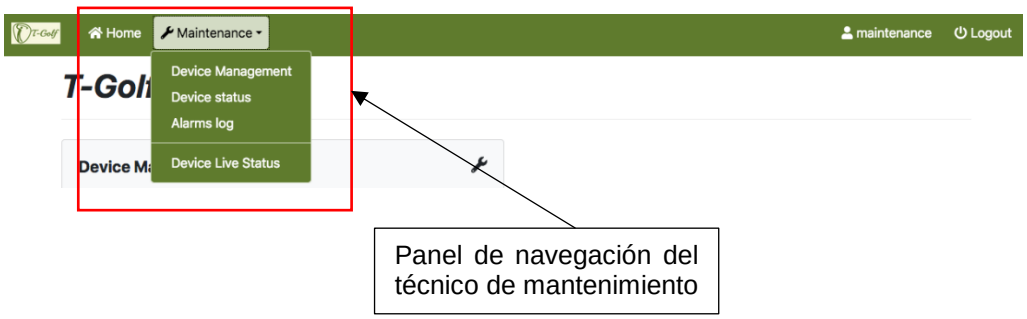
Los jugadores son los usuarios que tienen menos privilegios dentro de la aplicación (nivel de permiso *user*). Cuando un jugador está autenticado en la aplicación podrá acceder, como cualquier otro usuario, a los ajustes de su perfil, así como cerrar la sesión activa. Además de estas funciones, el jugador tendrá acceso a:

- Registro histórico de partidos jugados:
  - Detalle del partido seleccionado.
- Información sobre el partido en curso.

5.1.2 Técnico de mantenimiento

Los técnicos de mantenimiento (nivel de permiso *maintenance*) son los encargados de velar por el correcto funcionamiento de los dispositivos de medición instalados en los campos, por lo tanto, podrán visualizar su estado, así como gestionarlos. Estos usuarios tendrán acceso a:

- Gestión de los dispositivos:
  - Registrar nuevos dispositivos.
  - Editar o borrar dispositivos.
- Estado de los dispositivos.
- Histórico de los mensajes enviados por los dispositivos.
- Visión en tiempo real de la situación y estado de los dispositivos en el campo.



5.1.3 Caddie Master

Los caddie master (nivel de permiso *caddiemaster*) son los encargados de gestionar los caddies repartidos por el campo. Pero dado que en la actualidad hay cada vez menos caddies, su figura pasa a tener gran relevancia ya que tendrán que asegurarse que el tiempo que invierten los jugadores para completar un hoyo y, en consecuencia, el recorrido, no excedan de unos máximos establecidos por el campo y/o competición. Además, estos usuarios podrán dar de alta a los jugadores que participarán en los partidos, para poder monitorizarlos gracias a unos dispositivos de seguimiento. Los caddie master tendrán acceso a:

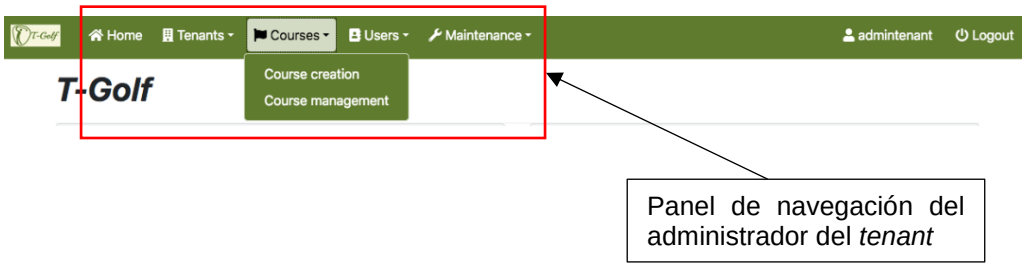
- Gestión de jugadores:
  - Alta de nuevos jugadores.
  - Visualización de jugadores dados de alta.
- Seguimiento de los jugadores participantes en un partido.

Panel de navegación del Caddie Master

5.1.4 Administrador del Tenant

Los administradores de los diferentes *tenant* (nivel de permiso *admintenat*) serán dados de alta por el administrador o administradores del sistema. Estos usuarios podrán editar la información de su *tenant* y gestionar los campos, usuarios y dispositivos. Los usuarios con este nivel de privilegios tendrán acceso a:

- Editar información de su *tenant*.
- Gestionar los campos de su *tenant*:
  - Dar de alta nuevos campos.
  - Editar o borrar campos existentes
- Gestionar los usuarios de su *tenant*:
  - Dar de alta nuevos usuarios. A excepción de jugadores u otros administradores de *tenant*.
  - Editar o borrar usuarios existentes.
- Gestión de los dispositivos:
  - Registrar nuevos dispositivos.
  - Editar o borrar dispositivos.
- Estado de los dispositivos.
- Histórico de los mensajes enviados por los dispositivos.
- Visión en tiempo real de la situación y estado de los dispositivos en el campo.

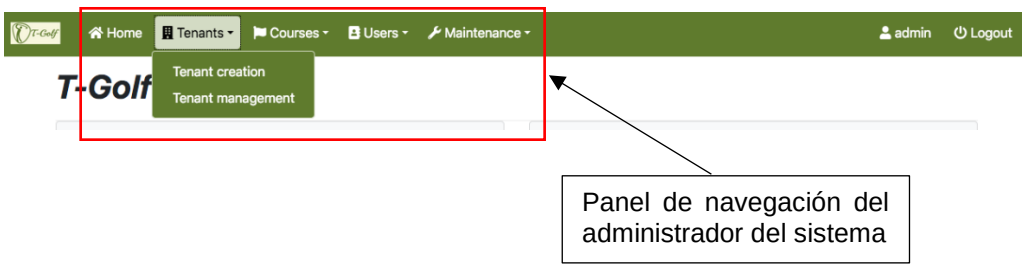


Los administradores de *tenant* tendrán acceso a la gestión y seguimiento del estado de los dispositivos de los campos ya que puede existir algún *tenant* en el que los técnicos de mantenimiento no sean los encargados de la monitorización de los dispositivos de lectura.

5.1.5 Administrador del sistema

Los administradores del sistema (nivel de permiso *admin*) son los usuarios con el máximo nivel de privilegios dentro de la aplicación. Estos usuarios serán similares a los administradores de *tenant*, pero serán los únicos que podrán gestionar cualquier tipo de usuario y tendrán acceso a la información y gestión de todos los *tenant* y campos del sistema. Los administradores del sistema podrán acceder a:

- Gestionar los *tenant* del sistema:
  - Dar de alta nuevos *tenant*.
  - Editar o borrar los *tenant* del sistema.
- Gestionar los campos del sistema:
  - Dar de alta nuevos campos.
  - Editar o borrar campos existentes
- Gestionar los usuarios del sistema:
  - Dar de alta nuevos usuarios. Son los únicos que pueden dar de alta a otros administradores del sistema y a los administradores de *tenant*.
  - Editar o borrar usuarios existentes.
- Gestión de los dispositivos:
  - Registrar nuevos dispositivos.
  - Editar o borrar dispositivos.
- Estado de los dispositivos.
- Histórico de los mensajes enviados por los dispositivos.
- Visión en tiempo real de la situación y estado de los dispositivos en el campo.



5.2 Login

El panel de inicio de la aplicación es una página de autenticación o de login. Cada vez que se acceda a la aplicación habrá que introducir las credenciales para poder acceder a cualquier sección. Desde la página de login los jugadores podrán registrarse en la aplicación, pero el resto de los usuarios tendrán que ser generados por los administradores o administradores de *tenant*.

Los usuarios se identifican en el sistema introduciendo el nombre de usuario, único dentro del sistema, y una contraseña asociada a este usuario. Si las credenciales introducidas no son correctas el sistema mostrará un mensaje de error y en la página aparecerán los botones de recuperación de usuario y/o contraseña.

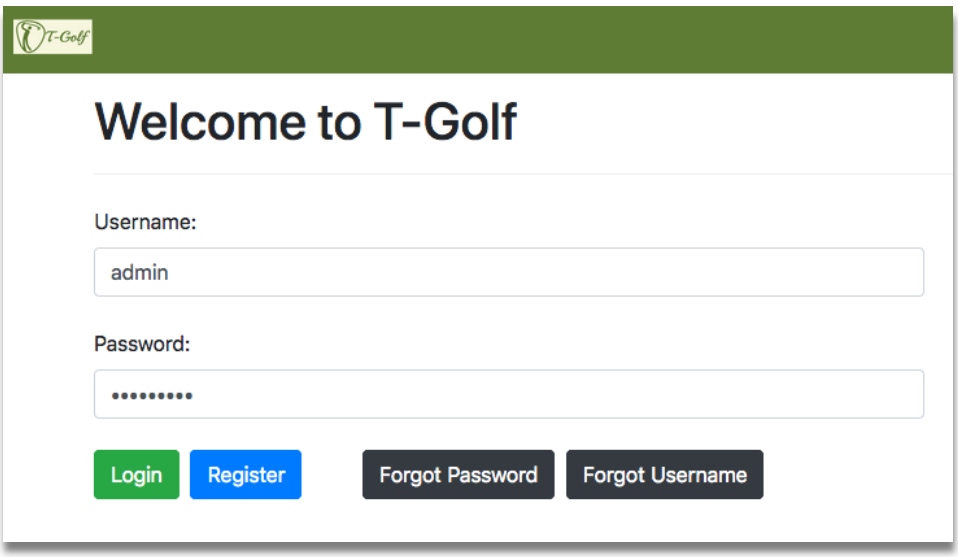


Figura 20. Botones de recordar contraseña y usuario activos

5.3 Recuperación de nombre de usuario y/o contraseña

Si el usuario no recuerda su nombre de usuario o la contraseña para poder identificarse en la aplicación, puede hacer clic en uno de los botones correspondientes que aparecen tras introducir incorrectamente las credenciales. Haciendo clic en cada uno de los botones al usuario se le solicitará información para poder enviar el dato olvidado por correo electrónico.

Si se quiere recuperar el nombre de usuario, el sistema solicitará el correo electrónico con el que se registró el usuario para enviar a esa dirección el nombre de usuario.

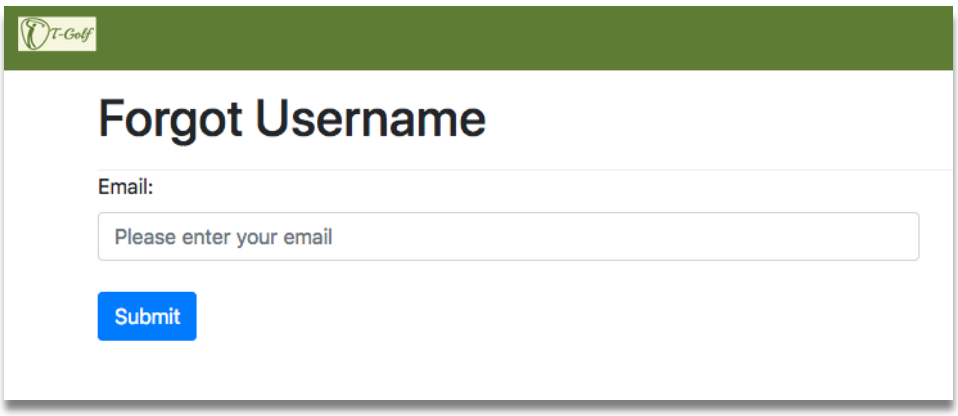


Figura 21. Pantalla de recuperación de nombre de usuario

Una vez introducido el correo electrónico de registro, el sistema enviará un mensaje a esa dirección con el nombre de usuario asociado a ese correo.

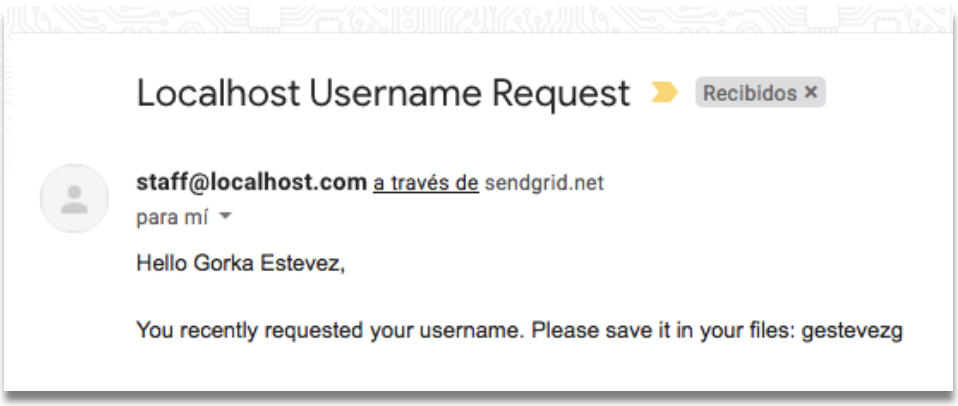


Figura 22. Correo de recuperación de usuario

Si el usuario no recuerda la contraseña, puede rellenar el formulario de recuperación de contraseña, que, en esta versión de la aplicación, solicita el nombre de usuario para enviar un enlace de restablecimiento de contraseña al correo electrónico. La contraseña no se envía al correo electrónico en texto plano dado que por razones de seguridad esta se guarda encriptada en la base de datos.

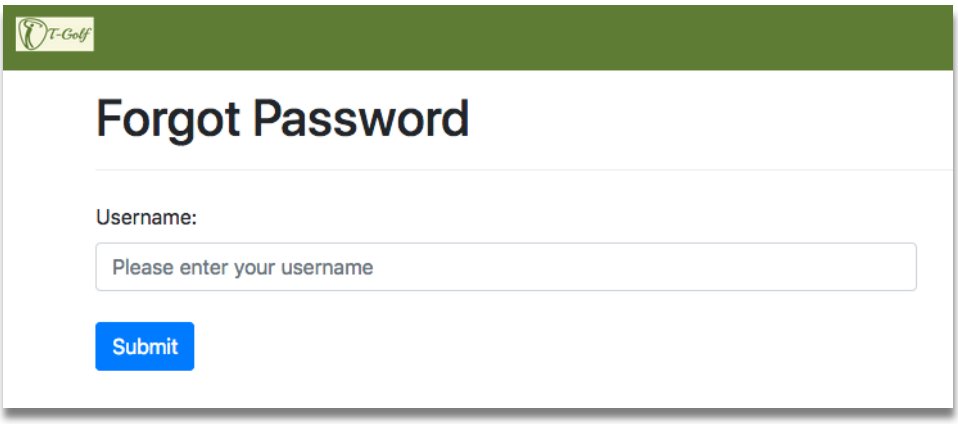


Figura 23. Pantalla de recuperación de contraseña

Cuando se solicita restablecer la contraseña, el correo que recibirá el usuario contendrá un enlace que le llevará a una página de la aplicación donde se podrá introducir una contraseña nueva.

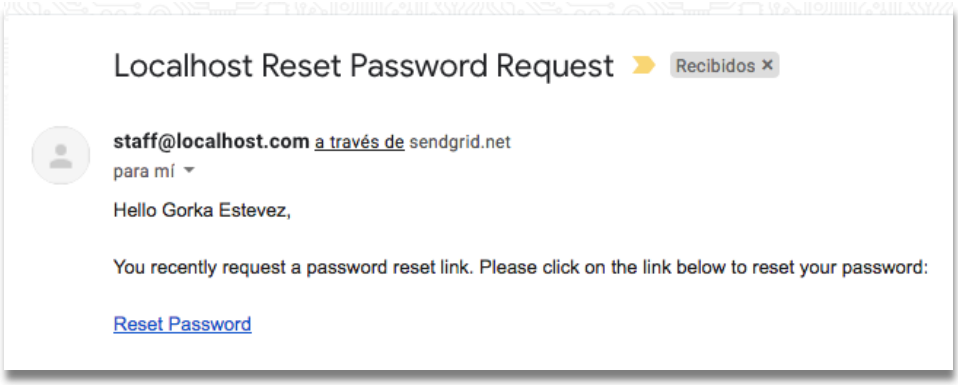


Figura 24. Correo de recuperación de contraseña

Tras introducir y confirmar la nueva contraseña, esta se guardará sustituyendo a la anterior en la base de datos y el usuario recibirá un correo indicando que la contraseña se ha restablecido.

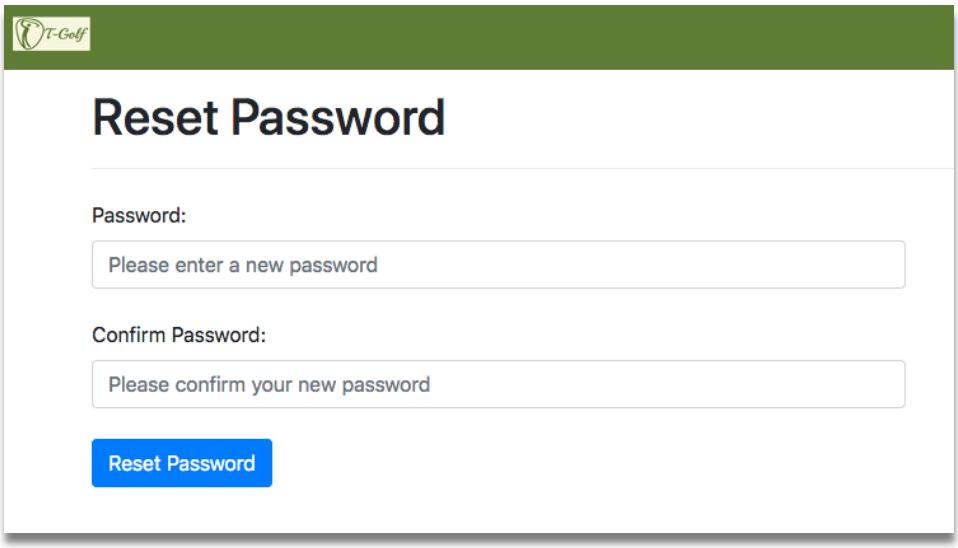


Figura 25. Pantalla de restablecimiento de contraseña

5.4 Perfil de usuario

Cualquier usuario autenticado en la aplicación podrá acceder a la información de su perfil haciendo clic en el nombre de usuario que aparece en la parte derecha del panel de navegación.

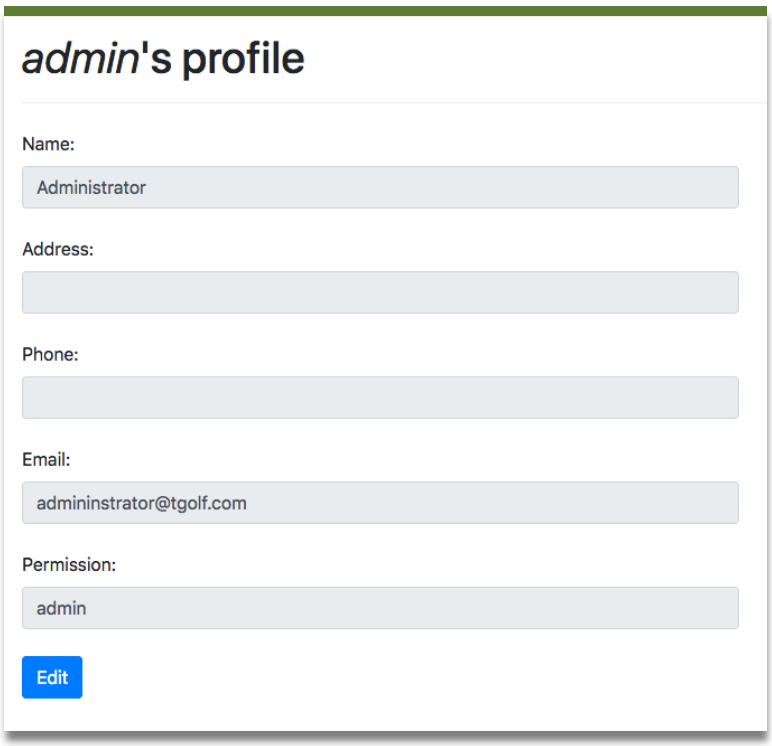


Figura 26. Pantalla de perfil de usuario

Dentro de la pantalla de perfil del usuario aparecen los datos relativos al usuario. Estos campos están bloqueados hasta que se hace clic en el botón editar, momento a partir del cual se pueden editar los campos: *Name*, *Address*, *Phone* e *Email*. El nivel del usuario no es modificable por el propio usuario y el nombre de usuario no es modificable en ningún caso ya que este identifica al usuario dentro del sistema.

Una vez pulsado el botón editar, aparece un nuevo botón que permitirá al usuario modificar la contraseña.

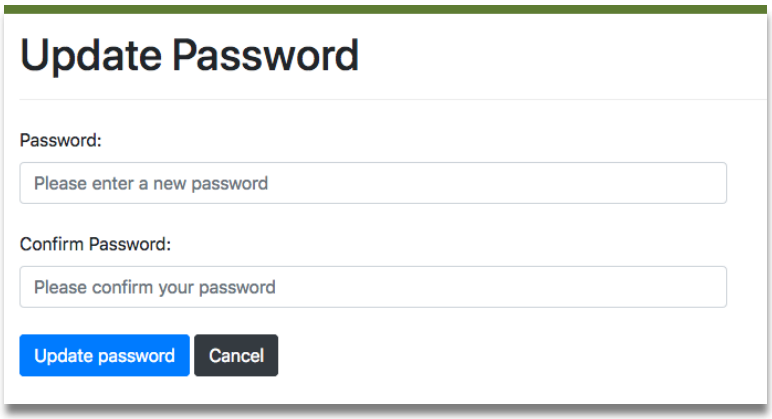


Figura 27. Pantalla de cambio de contraseña

5.5 Gestión de los *tenant*

La gestión de los *tenant* es la base para el funcionamiento de la aplicación. Si no se da de alta ningún *tenant* no podrán darse de alta campos ni usuarios, y sin los campos tampoco se podrán gestionar los dispositivos o los partidos.

Un *tenant* representa el nivel más alto en la estructura organizativa, es cualquier empresa encargada de gestionar uno o más campos de golf. Una vez se de alta un *tenant* se podrán dar de alta campos, dispositivos, partidos y usuarios.

Únicamente los administradores del sistema podrán dar de alta nuevos *tenant*. Para llevar a cabo un alta nueva se deberán dirigir a la opción *Tenant creation* del apartado *Tenants* del panel de navegación o haciendo clic en el botón *Register new tenant* en la pantalla *Home*.

En la pantalla de alta, el administrador deberá rellenar un formulario de datos, en los que habrá unos obligatorios, marcados con un asterisco, y otros opcionales. Así pues, los datos necesarios para dar de alta un *tenant* serán:

- Tenant ID: valor único que identificará la empresa en el sistema.
- CIF
- Phone
- Email

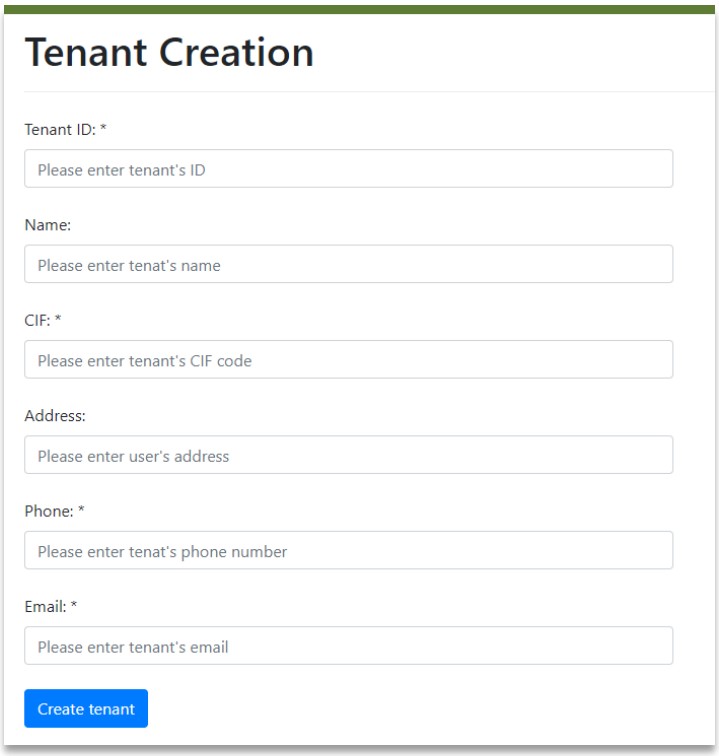


Figura 28. Formulario de creación de tenant

Es importante mencionar en este punto, que, si bien el administrador del sistema puede realizar las labores de un administrador de *tenant*, una buena práctica sería la de dar de alta un administrador de *tenant* para un correcto uso de la aplicación.

La segunda opción dentro de la sección *Tenants* permite al administrador del sistema gestionar las empresas dadas de alta. Cuando se accede a *Tenant management* se muestra una lista de las empresas dadas de alta y dos botones a la derecha de cada una de ellas que permiten editar sus datos (botón con el icono de un lápiz) o eliminarlos (botón con el icono de una papelera). Si se elimina un *tenant* se eliminarán todos los campos y usuarios asociados a este.

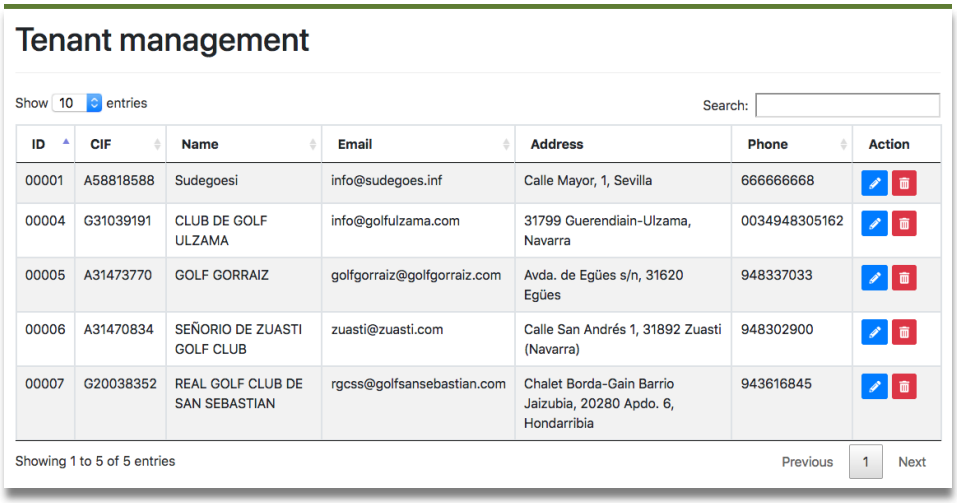


Figura 29. Tabla de tenants

En la figura anterior el administrador del sistema ha creado cinco empresas, y si quisiera editar cualquiera de ellas, deberá hacer clic sobre el botón azul con el icono de un lápiz. Tras pulsar dicho botón, la aplicación le mostrará una pantalla con los campos que se pueden editar.

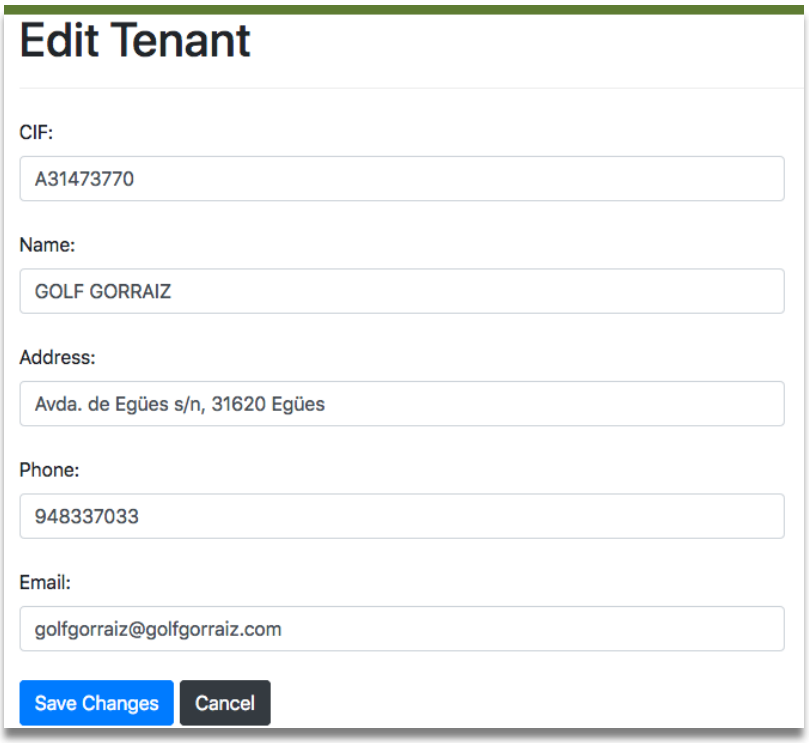


Figura 30. Formulario de edición de un tenant

5.6 Gestión de los campos

Como se ha visto al principio de la memoria, los *tenant* se componen de uno o más campos. El alta de un campo puede realizarlo un administrador del sistema o un administrador de *tenant*. La diferencia entre uno y otro es que los administradores del sistema podrán dar de alta campos en cualquier *tenant* del sistema mientras que los administradores de *tenant* solo podrán dar de alta campos en su *tenant*. Esta situación se ha tenido en cuenta a la hora de desarrollar la aplicación y en un caso se dará la opción de elegir el *tenant*, mientras que en el otro no.

De la misma manera que en el apartado anterior, a la hora de crear un campo, algunos datos serán obligatorios y otros opcionales. En este caso, los datos opcionales se auto rellenan con los datos del *tenant* al que pertenecerá el campo, permitiéndose su modificación.

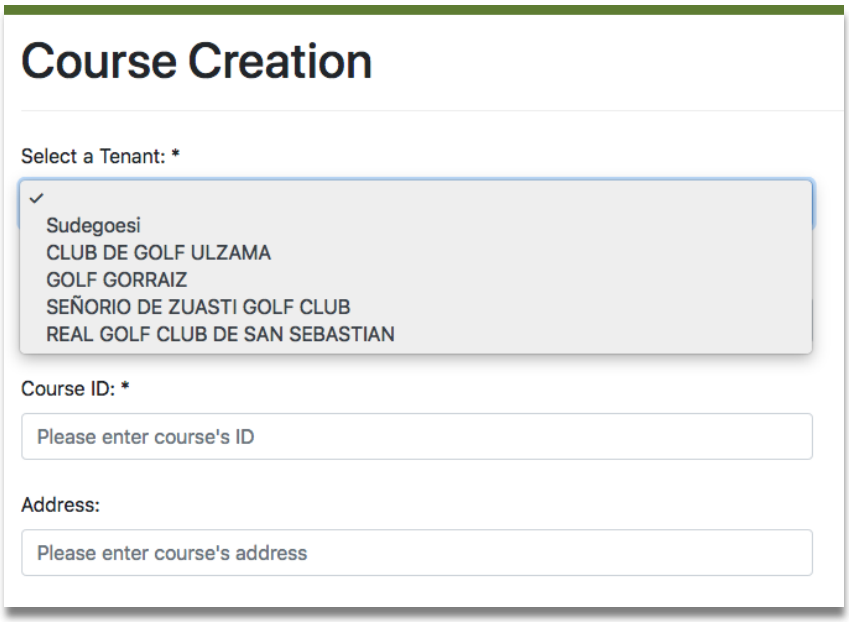


Figura 31. Creación de campo (administrador del sistema)

Los campos creados se pueden gestionar a través de la opción *Course management* de la sección *Courses* o desde la pantalla de inicio de la aplicación. Dentro de este apartado se mostrará un listado de los campos dados de alta y a la derecha de cada uno de ellos la posibilidad de editar o borrar el campo deseado. Si el usuario que accede a este apartado tiene permisos de administrador del sistema verá todos los campos dados de alta en el sistema y el *tenant* al que pertenecen, mientras que si el usuario tiene permisos de administrador de *tenant* únicamente verá los campos que pertenecen a su *tenant*.

5.7 Gestión de los usuarios

El apartado de gestión de se dividirá en función de los permisos de los usuarios que estén haciendo uso de la aplicación, ya que para diferentes niveles de permisos la aplicación mostrará unas opciones u otras.

5.7.1 Administrador del sistema

Los administradores del sistema podrán dar de alta a cualquier tipo de usuario y serán los únicos con nivel suficiente para poder dar de alta a otros administradores del sistema o a los administradores de *tenant*. Para dar de alta a un nuevo usuario, se puede seleccionar la opción *User creation* del apartado *Users* o pulsando el botón habilitado a tal efecto en la pantalla de inicio.

Cuando un administrador del sistema accede al formulario de creación de un nuevo usuario deberá rellenar todos los campos obligatorios, así como una contraseña. En el caso de crear un usuario de tipo jugador el *tenant* deberá ser el 99999 y si el usuario que se va a crear es otro administrador del sistema el *tenant* será el 00000. En cualquier caso, el sistema está diseñado para que, en el caso de no cumplirse estas reglas, estos usuarios especiales se guarden de forma correcta en la base de datos.

Una vez creado el usuario, este recibirá un correo electrónico en el que se le informa de la creación y de cuáles son sus credenciales de acceso a la plataforma. Dado que en esta primera versión de la aplicación la contraseña no se genera de forma aleatoria se recomienda al usuario que cambie la contraseña la primera vez que accede a la aplicación.

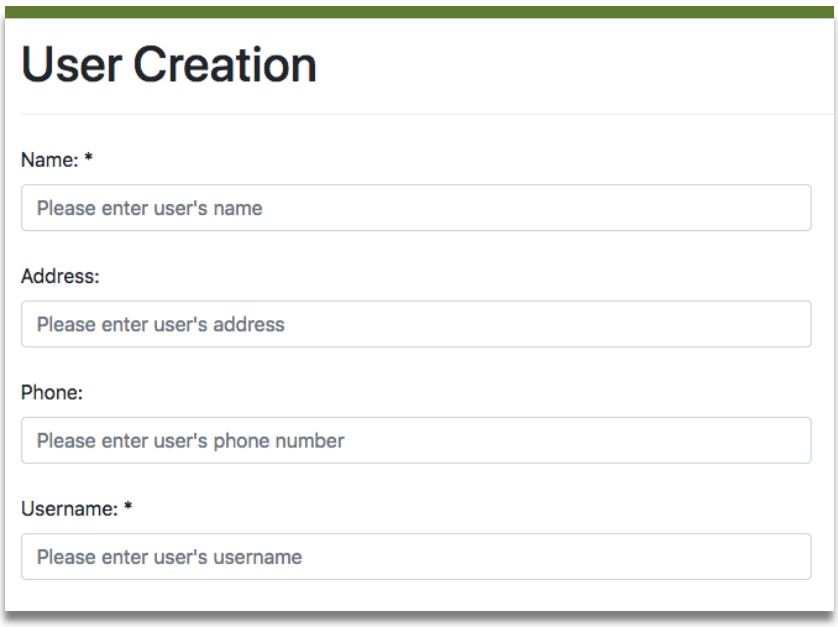


Figura 32. Parte del formulario de creación de usuarios

Al igual que con el resto de los datos almacenados en la base de datos, se puede obtener una relación de los usuarios dados de alta en el sistema haciendo clic en el botón de pantalla de inicio o a través de la opción *Users management* del apartado *Users* del panel de navegación.

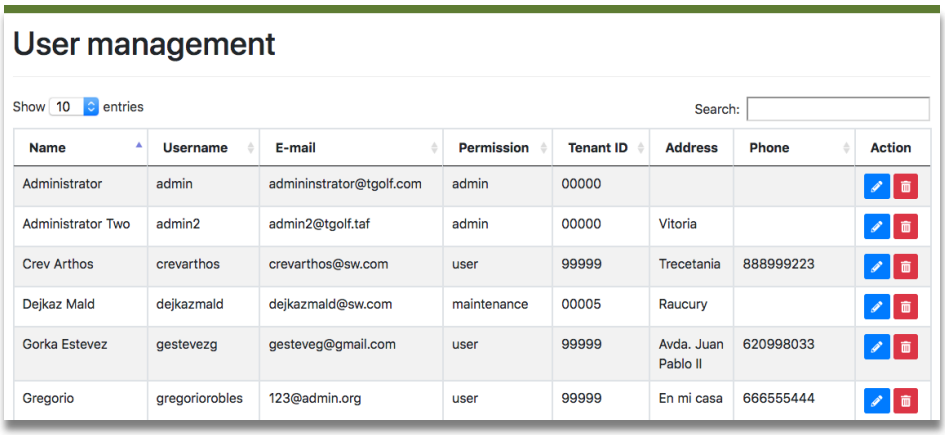


Figura 33. Listado de usuarios (administrador del sistema)

Desde esta pantalla se podrá eliminar cualquier usuario del sistema o editarlo. En este listado no aparece el usuario que está utilizando la aplicación ya que no tiene sentido que se pueda eliminar a sí mismo y la edición de sus datos se debe realizar desde el perfil de usuario. Para evitar borrados accidentales del usuario activo, la aplicación comprueba que el usuario que se vaya a borrar no sea dicho usuario.

Los administradores del sistema podrán editar todos los campos de un usuario a excepción de su contraseña y nombre de usuario, siempre que estén activados. En caso de no estarlo, la aplicación solicitará al administrador si desea activar el usuario. En la pantalla de edición de usuarios también se podrá modificar el nivel de permisos de los usuarios y el *tenant* al que pertenecen.

Los jugadores son usuarios especiales y tienen un tratamiento diferente al resto. Los administradores del sistema podrán crearlos y eliminarlos, pero nunca podrán modificar su *tenant* o nivel de permisos.

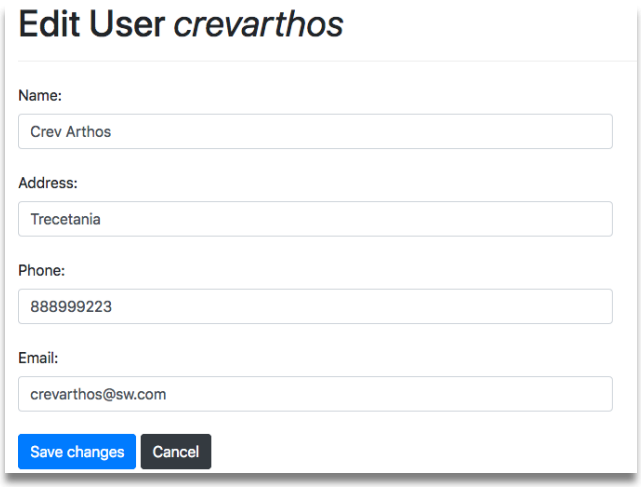


Figura 34. Edición de los datos de un jugador

5.7.2 Administrador del tenant

Los administradores de los *tenant* podrán gestionar los usuarios de forma similar a como los gestionan los administradores del sistema. Desde el apartado *Users*, podrán dar de alta usuarios y gestionarlos con las limitaciones propias de su nivel de permisos.

La pantalla de creación de usuarios ofrecerá los mismos campos que en el caso de los administradores del sistema a excepción del *tenant*, ya que los administradores de *tenant* solo pueden generar usuarios dentro de su *tenant*, y la elección de permisos se recorta a usuarios de tipo *caddie master* o *maintenance*. Como en el caso de los administradores de sistema, una vez creado el usuario, este recibirá un correo con las credenciales para poder acceder a la aplicación y un aviso para que cambie la contraseña la primera vez que acceda.

La pantalla de gestión de usuarios dados de alta en el *tenant* es idéntica a la que ven los administradores del sistema, pero en este caso, se omite la columna *Tenant ID* y no se muestran los jugadores. Estos últimos solo son visibles por parte de los administradores del sistema o los *caddie master*.

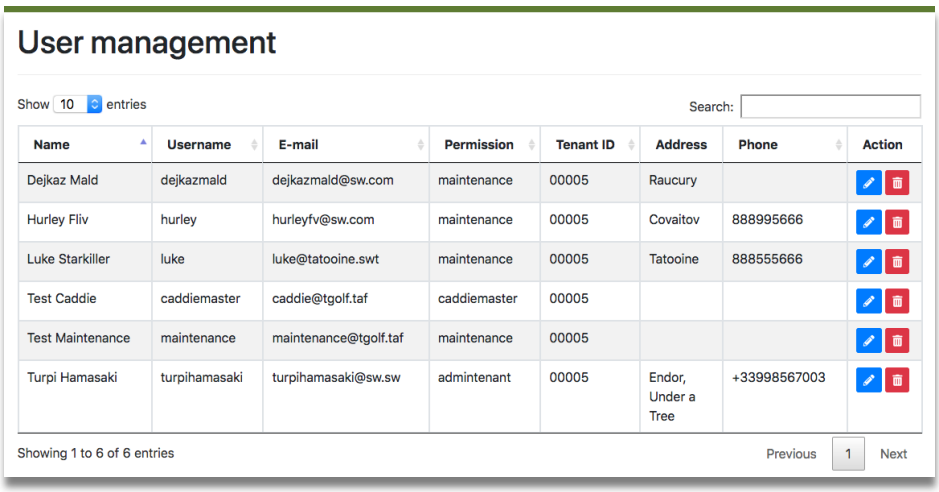


Figura 35. Listado de usuarios (administrador de tenant)

Como se puede ver en la figura, desde esta pantalla se puede editar o borrar cualquier usuario. La edición de usuarios permite a los administradores de los *tenant* modificar el nivel de permiso de un usuario entre *caddie master* o *maintenance*.

5.7.3 Caddie Master

En el caso de los caddie master la nomenclatura de la sección de usuarios en el panel de navegación y en los botones de la pantalla de inicio cambia para adaptarse a este tipo de usuario.

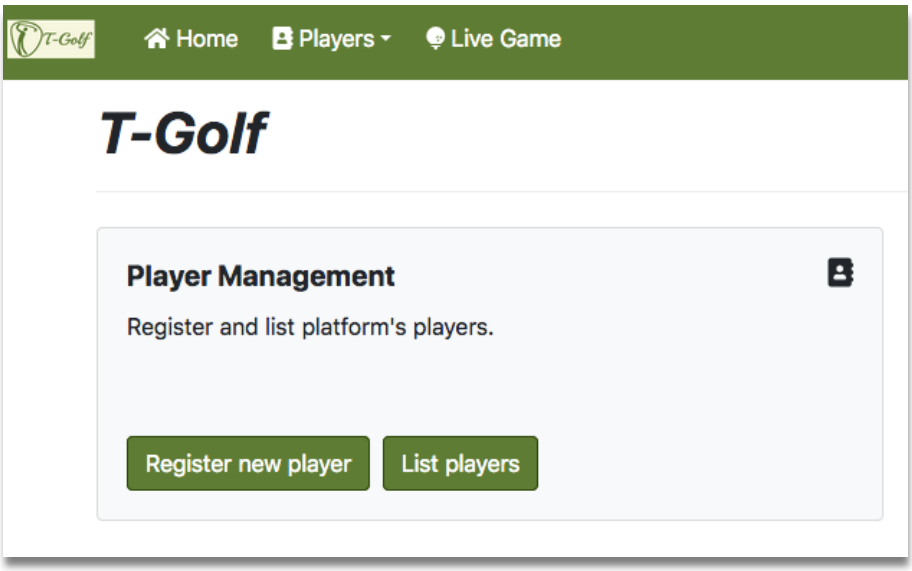


Figura 36. La sección Users pasa a llamarse Players

Como se aprecia en la figura, la sección para gestionar los usuarios pasa a llamarse *Players* cuando se autentica un *caddie master*. Esto es debido a que este tipo de usuarios solo podrán dar de alta jugadores y únicamente podrán ver a los jugadores registrados en el sistema.

La pantalla de registro de nuevos jugadores es similar a la que se muestra a los administradores del sistema o de los *tenant*. Pero en el caso de los *caddie master* no se permitirá elegir el nivel de permiso ni el *tenant* al que pertenecerá el usuario creado. Los usuarios generados por los *caddie master* siempre serán jugadores por lo que el nivel de permiso será *user* y el *tenant* el 99999.

Otro cambio que se observa es en el listado de usuarios, donde desaparece la columna *Action*, ya que no se permite que los *caddie master* editen o eliminen usuarios.

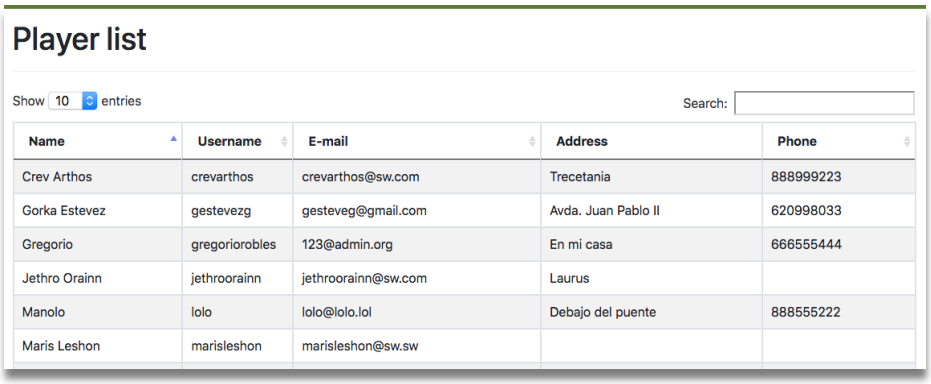


Figura 37. Listado de jugadores (caddie master)

5.7.4 Jugadores

Cualquier jugador podrá registrarse en la aplicación desde la pantalla de login. En esta pantalla existe un botón (*Register*) el cual lleva a una pantalla de registro donde los jugadores podrán introducir sus datos y enviarlos al sistema.

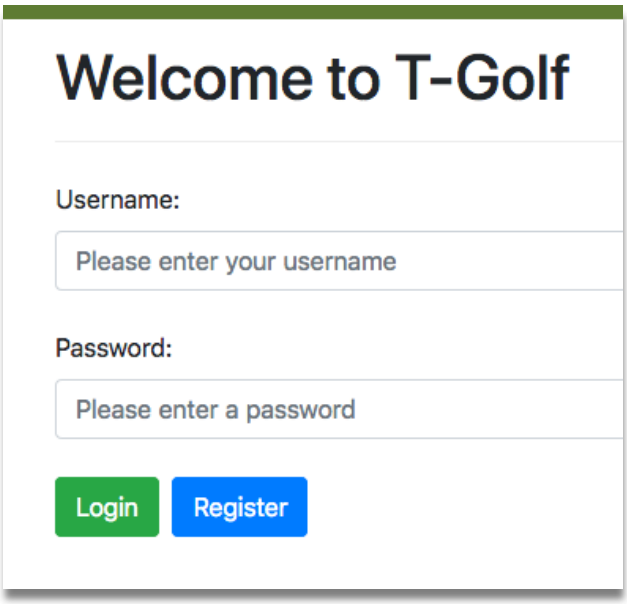


Figura 38. En azul, el botón de registro

Una vez se ha rellenado el formulario de registro y se ha enviado, el usuario recibirá un correo electrónico confirmándole el correcto registro en la aplicación y la necesidad de activar su cuenta a través de un enlace proporcionado en el mismo mensaje.

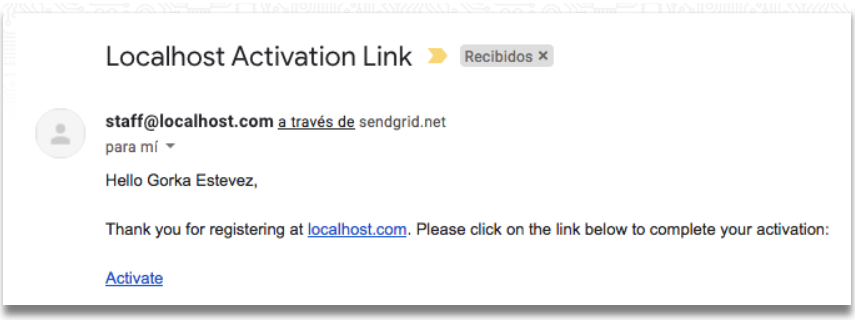


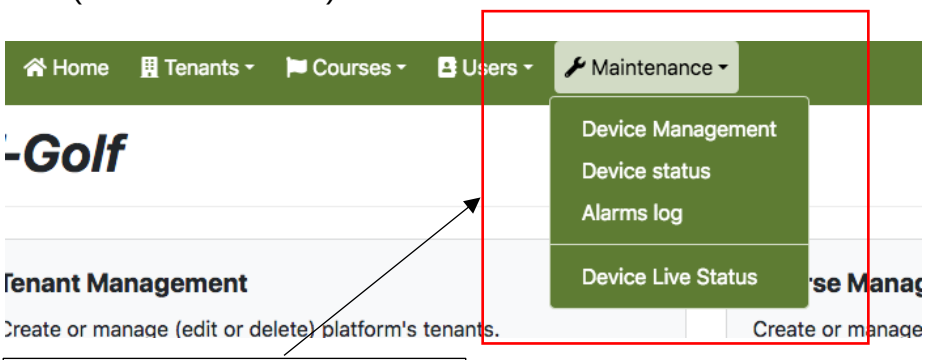
Figura 39. Mensaje de activación de la cuenta

5.8 Gestión de los dispositivos

Una de las funcionalidades que se ha desarrollado en la aplicación es la posibilidad de acceder a la información que recogen los diferentes dispositivos instalados en los campos de golf.

Estos dispositivos se conectan a la base de datos y donde van guardando la información en tiempo real. La conexión entre los dispositivos y la base de datos no entra dentro del ámbito del desarrollo de esta aplicación ya que utilizan un sistema independiente a esta. Una vez almacenada la información es recogida por la aplicación para poder ser accesible por los administradores del sistema, de los *tenant* y los técnicos de mantenimiento.

Para acceder a la información de estos dispositivos se ha diseñado una sección específica denominada *Maintenance* que se divide en cuatro apartados: gestión de los dispositivos (*device management*), estado de los dispositivos (*device status*), log de alarmas (*alarms log*) y situación den vivo de los dispositivos (*device live status*).



Sección de mantenimiento, desde donde se pueden gestionar los dispositivos.

Dentro de *Device Management* se podrá ver una lista con los dispositivos actualmente dados de alta en el sistema. En el caso de los administradores del sistema en la lista aparecerán todos los dispositivos, y si el usuario activo es un administrador de *tenant* o un técnico de mantenimiento aparecerán los dispositivos de los campos del *tenant* al que pertenezcan los usuarios.

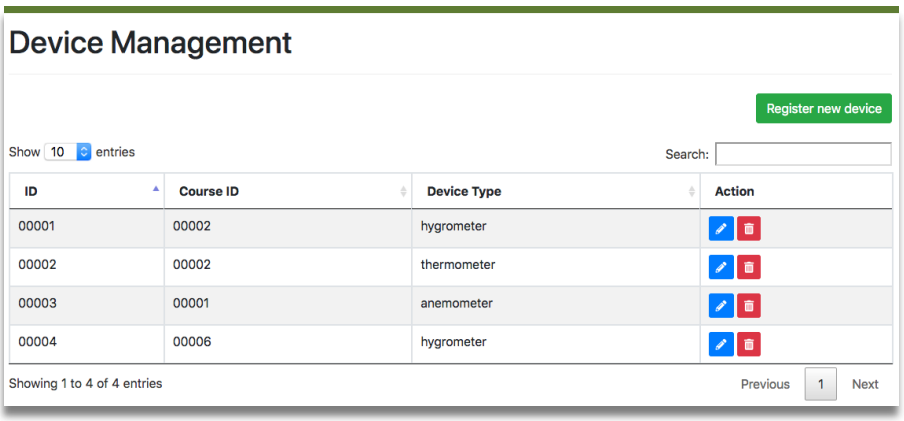


Figura 40. Listado de dispositivos

Como se puede observar en la figura 26, desde esta pantalla se puede acceder a la pantalla de registro de un nuevo dispositivo, así como editar o borrar el dispositivo deseado. Tal y como está desarrollado el sistema, la eliminación de un dispositivo de la aplicación es meramente lógica, desaparece la entrada de la base de datos, pero el dispositivo continúa funcionando ya que no se actúa sobre este.

Si se desea dar de alta un nuevo dispositivo en el sistema, la aplicación nos solicitará el campo donde va a estar instalado, su identificador y el tipo de dispositivo. Para el desarrollo de la aplicación el identificador del dispositivo es un número cualquiera de cinco cifras, pero si la aplicación se instala en un entorno real, este identificador tendrá que coincidir de alguna manera con algún identificador único del dispositivo (un número de serie o una dirección MAC, por ejemplo).

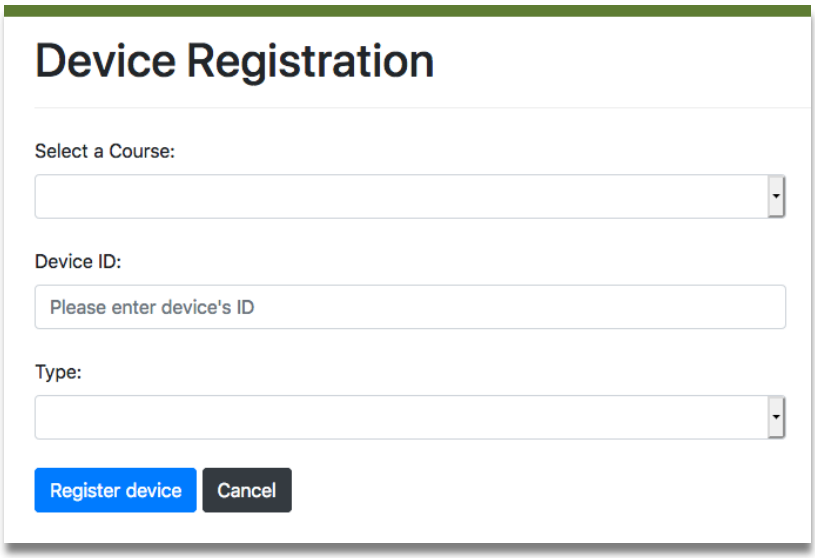


Figura 41. Pantalla de registro de un dispositivo

Un administrador del sistema podrá dar de alta un dispositivo en cualquier campo del sistema, mientras que a los administradores de los *tenant* o a los técnicos de mantenimiento solo se les ofrecerán los campos de su *tenant*.

En la pantalla de edición de un dispositivo, se le podrá asignar a otro campo o cambiarlo de tipo.

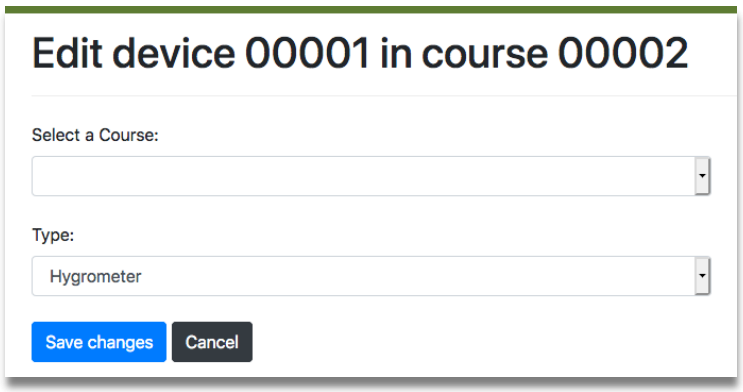


Figura 42. Pantalla de edición de un dispositivo

En el apartado *Device status* se mostrará una tabla con la situación de los dispositivos, incluyendo el identificador del dispositivo, el campo donde está instalado, el tipo, su estado y el último mensaje guardado en la base de datos.

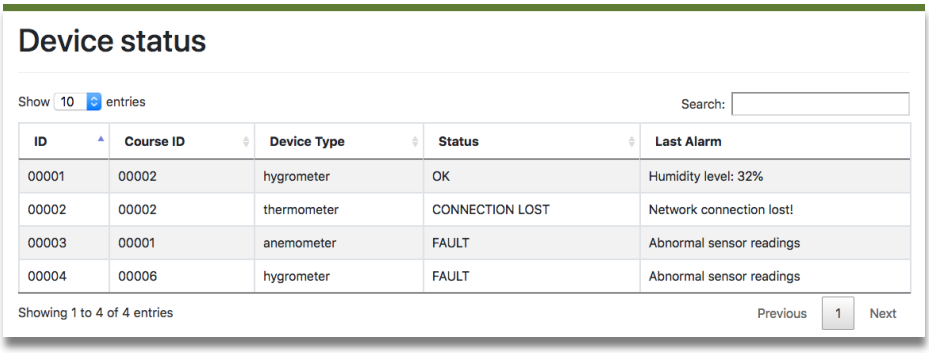


Figura 43. Estado de los dispositivos

Cada dispositivo podrá tener tres estados (ampliables en un futuro):

- OK: el dispositivo está conectado al sistema y funciona correctamente. En este estado el dispositivo envía mensajes con las lecturas que efectúa.
- FAULT: el dispositivo está conectado al sistema, pero presenta algún fallo de funcionamiento. El dispositivo envía un mensaje con información sobre el problema.
- CONNECTION LOST: el dispositivo no está conectado con el sistema.

En la pantalla anterior, la aplicación muestra para cada dispositivo el último mensaje registrado en la base de datos. Pero la aplicación permite ver el histórico de mensajes de todos los dispositivos en la sección *Alarmas log*. En esta pantalla, se muestra una tabla ordenada por fecha y hora en orden descendente con los mensajes de todos los dispositivos.

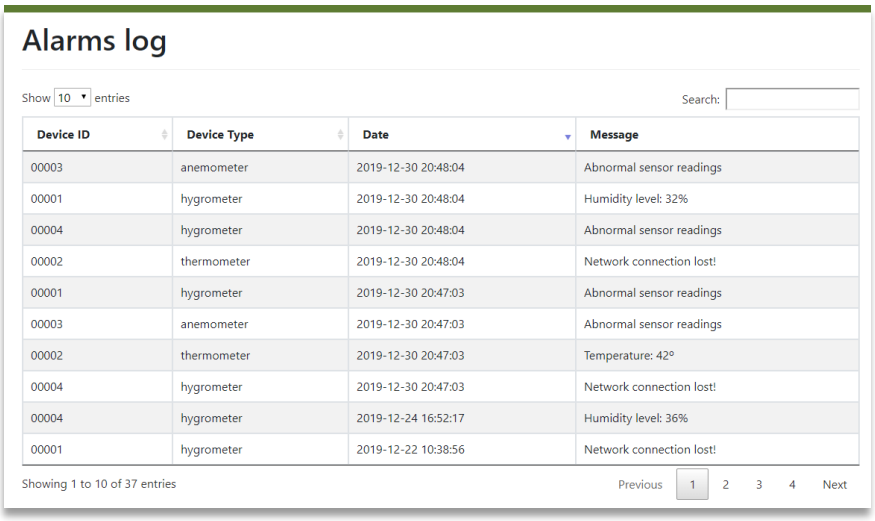


Figura 44. Tabala de alarmas (mensajes)

Para terminar, esta sección tiene una utilidad más, la cual permite observar los dispositivos en el campo y conocer su estado en tiempo real. Entrando en *Device Live Status* se puede elegir un campo y ver la ubicación de los dispositivos, así como su estado.

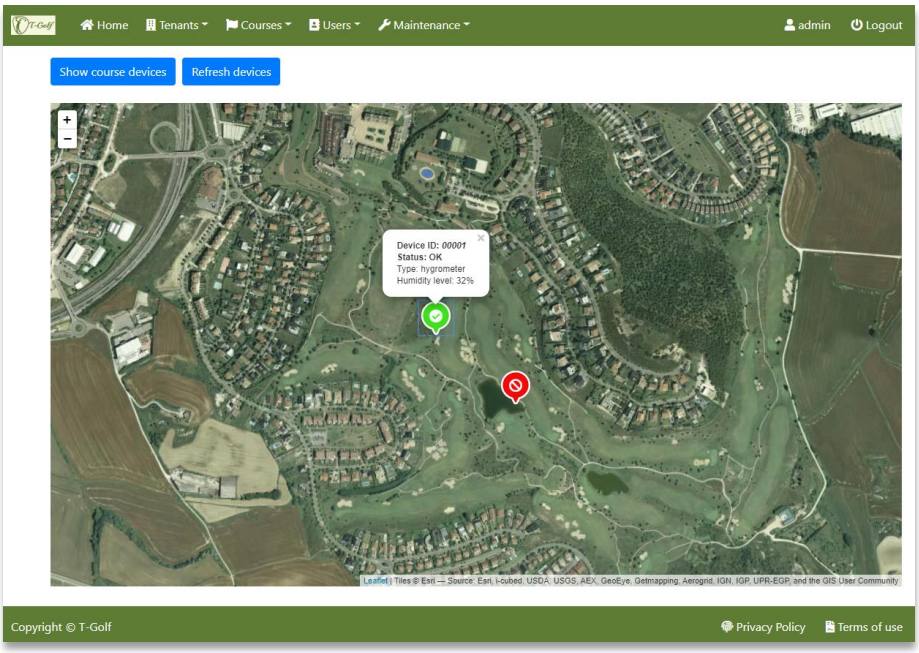


Figura 45. Estado de los dispositivos en tiempo real

En el mapa del campo seleccionado, los dispositivos aparecen representados con marcadores cuyo color permite conocer el estado de un dispositivo de un solo vistazo: verde para el estado OK, amarillo para el estado FAULT y rojo para el estado CONNECTION LOST. Además, haciendo clic en los marcadores se muestra en un bocadillo la información del dispositivo. Esta información es la misma que la de la tabla *Device status,* pero presentada de una forma más visual. Si se quiere refrescar la situación de los dispositivos, bastaría con pulsar el botón *Refresh devices*.

5.9 Seguimiento del juego

Uno de los mayores problemas a la hora de gestionar los partidos de golf es el tiempo que necesitan los jugadores para completar un hoyo o el recorrido entero. En las competiciones más importantes de Europa este tiempo está tipificado en el reglamento, pero no es lo habitual. Es por ello, que los *caddie master* se ven en la necesidad de ordenar a los jugadores en función de su tiempo de juego, por ejemplo, permitiendo pasar a aquellos jugadores que se están viendo retrasados por un grupo de jugadores más lentos. Estas decisiones son difíciles de tomar ya que tener un control de todos los jugadores dentro de un campo de golf es una tarea muy complicada.

Para facilitar esta tarea la aplicación pone a disposición de los *caddie master* la posibilidad de monitorizar la situación de los jugadores sobre el campo en tiempo real. Para de esta forma poder tomar las decisiones pertinentes con aquellos jugadores que estén ralentizando el juego.

Para acceder a esta funcionalidad bastará con hacer clic en *Live Game* en el panel de navegación o en el botón habilitado en la pantalla de inicio. Una vez en la pantalla de *Live Game* se seleccionará un campo y se podrá observar el desarrollo del juego.

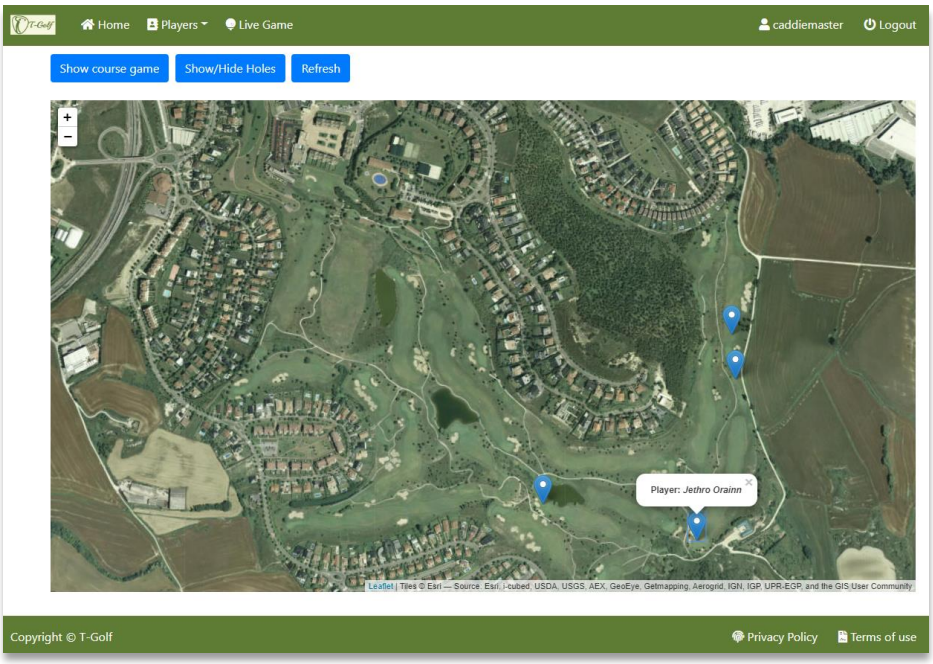


Figura 46. Jugadores en tiempo real

La figura 31 muestra la situación en tiempo real de cuatro jugadores en uno de los campos de golf registrados en el sistema. Esta pantalla se actualiza cada minuto, pero también se puede forzar la actualización de la vista haciendo clic en el botón *Refresh*.

Además de la situación de los jugadores en el campo, haciendo clic en el botón *Show/Hide Holes* se muestran y ocultan los hoyos del campo. Y haciendo clic en cualquiera de ellos se puede ver el número de hoyo, así como su par.

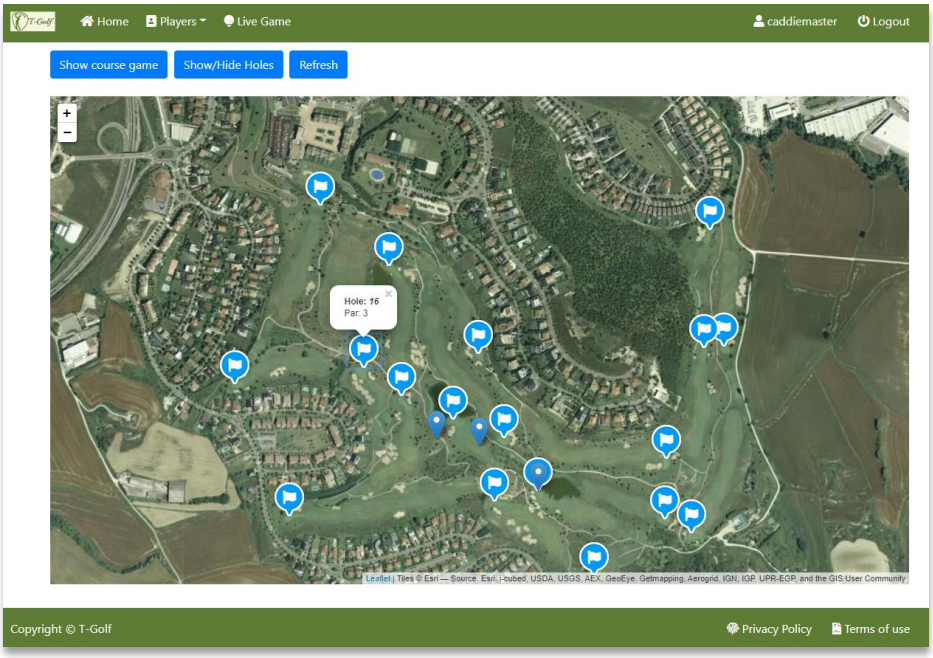


Figura 47. Situación de los hoyos en el campo

5.10 Estadísticas de los jugadores

La aplicación pretende ofrecer un valor añadido a los jugadores registrados en la misma y por ello se les permitirá visualizar las estadísticas de los partidos que vayan completando y del partido en juego. Estos datos serán almacenados en la base de datos bien por parte del jugador, introduciéndolos en unos dispositivos que se les ofrecerán, o por parte de la organización del campeonato o partido.

Cuando un jugador accede a la aplicación podrá ver el registro histórico de partidos haciendo clic en *Historic Games Record* del panel de navegación o en el botón habilitado en su pantalla de inicio.

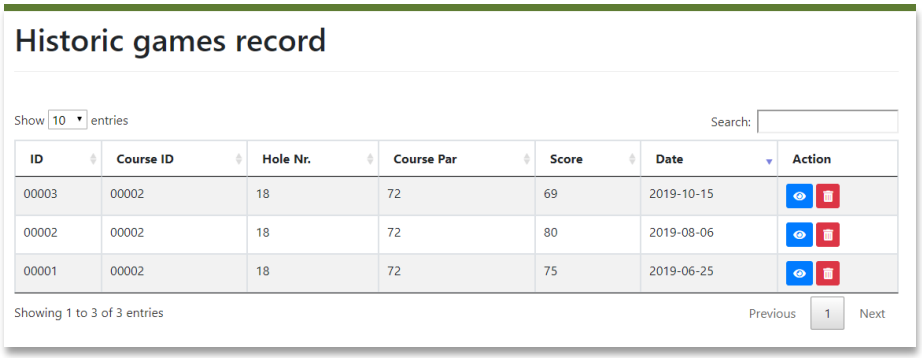


Figura 48. Registro histórico de partidos

En este registro los jugadores podrán eliminar los partidos que deseen pulsando el botón rojo con un icono de una papelera o ver los detalles del partido pulsando el botón azul con el icono de un ojo.

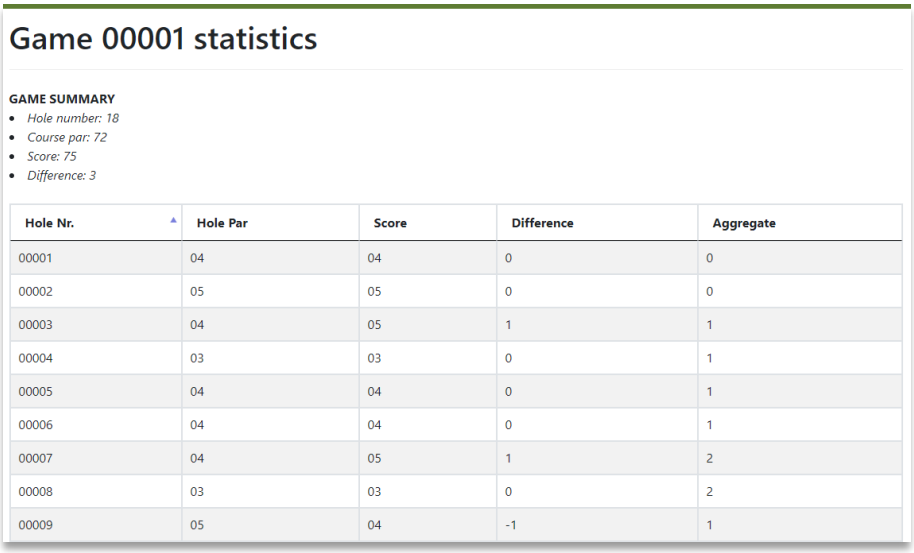


Figura 49. Detalle de un partido

En la pantalla de detalle el jugador podrá ver un resumen con los datos del partido y una tabla donde aparece el par de cada hoyo, así como el número de golpes realizados por el jugador. Esta información se complementa con la diferencia entre el par del hoyo y el resultado del jugador y el acumulado de la diferencia en cada hoyo.

6. Conclusiones

Puede parecer obvio mencionarlo, pero lo que he aprendido gracias a este trabajo, básicamente, ha sido desarrollar una aplicación web. Sin embargo, esto conlleva muchas cosas. He aprendido a hacerlo utilizando el método “MEAN stack”, es decir, la combinación de mongoDB, Express, Angular y Node. El aprendizaje ha ido más allá, porque debería mencionar también la planificación del trabajo, la consecución de los objetivos o la gestión del tiempo para cumplir los hitos. Aunque tenía clara la idea de cómo quería que fuese el resultado final de la aplicación, durante el proceso de elaboración, he visto que van apareciendo más flecos y que debía estar continuamente revisando la planificación realizada, haciendo constantes evaluaciones del progreso. Esa ha sido la mayor lección aprendida. Sin embargo, debo subrayar que el seguimiento de la metodología prevista ha sido la adecuada y no ha habido que introducir cambios significativos para garantizar el éxito del trabajo. Los cuatro hitos establecidos te permiten un pequeño margen flexible en este sentido. Por otro lado, es verdad que la idea inicial se ha modificado algo. Se planteó el proyecto como el desarrollo de una aplicación web para la gestión del eficiente del juego en los campos de golf y se ha desarrollado una para la gestión global de los campos. Esto se debe a que en la revisión del objetivo se vio que, para llegar a una gestión del juego, era necesario previamente una gestión del propio campo. Por falta de tiempo, me ha quedado por profundizar en CSS, es decir, en los estilos HTML. A futuro, las mejoras que podrían hacerse sería la de “pintar” los hoyos en los mapas o el de definir un sistema de gestión de alarmas para que se puedan personalizar los avisos de los dispositivos.

7. Glosario

Aplicación web: herramienta que los usuarios pueden utilizar accediendo a un servidor web a través de internet o de una intranet mediante un navegador.

Atom: editor de código fuente de código abierto para macOS, Linux, y Windows con soporte para múltiples plugin escritos en Node.js y control de versiones Git integrado, desarrollado por GitHub.

Backend: la parte de un sistema informático o aplicación a la que no accede directamente el usuario, normalmente responsable del almacenamiento y manipulación de datos.

Bootstrap: entorno de trabajo CSS y JavaScript diseñado para la creación de interfaces limpias y con diseño adaptable.

Caddie Master: persona encargada de organizar y formar a los caddies de los clubes, además los distribuye a los jugadores según sus características. También se encarga de organizar el parque de carros y encargar que limpien los palos, las bolsas y los *buggies*. En los campos donde no haya *caddies*, se ocupan de todo lo relativo al funcionamiento del campo de golf.

Contendor principal: zona de la aplicación donde se muestra la información solicitado por el usuario.

Controlador: código que se encarga de unir las vistas con el modelo.

CSS: lenguaje de diseño gráfico que permite definir y crear la presentación de un documento escrito en algún lenguaje de marcado.

Datatables: plugin de código abierto para la librería JavaScript de jQuery que permite renderizar tablas HTML con funcionalidades avanzadas.

EIBT: Empresa Innovadora de Base Tecnológica.

Express: el entorno de trabajo web más popular de Node.

Frontend: relacionado o que denota la parte de un sistema informático o aplicación con la que el usuario interactúa directamente.

Gestor de paquetes: colección de herramientas que sirven para automatizar el proceso de instalación, actualización, configuración y eliminación de paquetes de software.

Green: zona en la que se encuentra el hoyo donde debe entrar la bola.

Hoyo: en el contexto de este TFG puede referirse a la parte del campo que delimita el área de juego desde el *tee* de salida hasta la bandera, o al hoyo físico donde se debe de introducir la bola de golf.

HTML5: quinta versión del lenguaje básico de la World Wide Web.

jQuery: es una biblioteca de JavaScript que permite simplificar la manera de interactuar con documentos HTML, manejar eventos, manipular el árbol DOM, desarrollar animaciones y agregar interacción con la técnica AJAX.

Leaflet: librería JavaScript de código abierto diseñada para crear mapas interactivos.

Login: autenticarse en la aplicación.

MongoDB: sistema de gestión de bases de datos NoSQL distribuido de tipo documental. Especialmente adecuado para la creación de aplicaciones web.

Mongoose: Mongoose es una herramienta de modelado de objetos MongoDB diseñada para trabajar en un entorno asíncrono.

Multi-tenant: principio de arquitectura de software en la cual una sola instancia de la aplicación se ejecuta en el servidor, pero sirviendo a múltiples clientes u organizaciones (*tenant*).

Node: entorno de ejecución JavaScript diseñado para la creación de aplicaciones de red escalables.

NPM: Node Packet Manager. Gestor de paquetes de Node.

Panel de navegación: zona de la aplicación donde se muestran las acciones más relevantes para el usuario.

Par: número predeterminado de golpes que un jugador complete un hoyo, un recorrido (la suma de los pares de hoyos jugados) o un torneo (la suma de los pares de cada recorrido).

Pyme: Empresa mercantil, industrial, etc., compuesta por un número reducido de trabajadores, y con un moderado volumen de facturación.

Servicio: función que se utiliza en los controladores y la ruta que se utilizará en el archivo api.js para obtener la respuesta.

Single-Page Application: aplicación web que se ejecuta bajo una única página.

Tenant: organización o empresa que hace uso de la aplicación.

Usuario: cualquier persona que esté registrada en el sistema y utilice la aplicación.

Vista: página o contenido que se le muestra al usuario al utilizar la aplicación.

8. Bibliografía

[1] MDN. (5 de enero de 2020). *Express/Node introduction*. Recuperado en octubre de 2019, de developer.mozilla.org: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction

[2] W3Schools. (2019). *JavaScript Tutorial*. Recuperado en octubre de 2019, de w3schools: https://www.w3schools.com/js/default.asp

[3] Paramio, C. (10 de mayo de 2011). *Una introducción a MongoDB*. Recuperado en octubre de 2019, de genbeta.com: https://www.genbeta.com/desarrollo/una-introduccion-a-mongodb

[4] MDBootstrap. (2019). *Introduction to MongoDB*. Recuperado en octubre de 2019, de docs.mongodb.com: https://docs.mongodb.com/manual/introduction/

[5] Manuel, A. J. (3 de septiembre de 2014). *Fundamentos de bases de datos NoSQL: MongoDB*. Recuperado en Octubre de 2019, de campusmvp.es: https://www.campusmvp.es/recursos/post/Fundamentos-de-bases-de-datos-NoSQL-MongoDB.aspx

[6] Basalo, A. (28 de agosto de 2014). *Qué es Angular JS.* Recuperado en octubre de 2019, de desarrolloweb.com: https://desarrolloweb.com/articulos/que-es-angularjs-descripcion-framework-javascript-conceptos.html

[7] W3Schools. (2019). *AngularJS Tutorial*. Recuperado en octubre de 2019, de w3schools.com: https://www.w3schools.com/angular/default.asp

[8] David. (7 de julio de 2017). *Explicación del patrón MVC en AngularJS*. Recuperado en octubre de 2019, de guidacode.com: https://www.guidacode.com/2017/angularjs/explicacion-del-patron-mvc-en-angularjs/

[9] W3Schools. (2019). *HTML Tutorial*. Recuperado en octubre de 2019, de w3schools.com: https://www.w3schools.com/html/default.asp

[10] W3Schools. (2019). *jQuery Tutorial*. Recuperado en octubre de 2019, de w3schools.com: https://www.w3schools.com/jquery/

[11] Gustavo. (13 de mayo de 2019). *Qué es jQuery*. Recuperado en octubre de 2019, de hostinger.es: https://www.hostinger.es/tutoriales/que-es-jquery/

[12] Axarnet. (01 de diciembre de 2019). *Bootstrap - ¿Qué es y Cómo funciona?* Recuperado en diciembre de 2019, de axarnet.com: https://axarnet.es/blohttps://datatables.net/g/bootstrap

[13] Rodriguez, T. (16 de junio de 2012). *Bootstrap*. Recuperado en octubre de 2019, de genbeta.com: https://www.genbeta.com/desarrollo/bootstrap

[14] Agafonkin, V. (2019). *Leaflet API reference*. Recuperado en diciembre de 2019, de leafletjs.com: https://leafletjs.com/reference-1.6.0.html

[15] SpryMedia Ltd. (2019). *DataTables*. Recuperado en noviembre de 2019, de datatables.net: https://datatables.net/

[16] *Getting Started*. (s.f.). Recuperado en octubre de 2019, de mongoosejs.com: https://mongoosejs.com/docs/index.html

9. Anexos

9.1 Anexo I

Este anexo sirve como breve manual para arrancar el servidor y los dos simuladores de dispositivos que se han programado.

En la raíz del archivo comprimido *tgolf_app.zip* se encuentran las carpetas *tgolf_app* y *device_simulator*. El servidor se arranca ejecutando el comando *node server.js* en la carpeta *tgolf_app*. Tras ejecutar ese comando se recibirá un mensaje indicando que la conexión con la base de datos se ha realizado correctamente y que el servidor está escuchando en el puerto 8000.

```
Running the server on port 8000
Succesfully connected to MongoDB
```

Una vez el servidor haya arrancado se puede acceder a la aplicación introduciendo la dirección http://localhost:8000. A continuación, se indican unos usuarios de prueba generados:

- usuario: user contraseña: User123.
- usuario: maintenance contraseña: Maint123.
- usuario: caddiemaster contraseña: Caddie123.
- usuario: admintenant contraseña: Admint123.
- usuario: admin2 contraseña: Admin123.

En la carpeta *device_simulator* se encuentran los archivos *deviceServe.js* y *livegame.js*. El primero de ellos cambia el estado y los mensajes de los dispositivos instalados en el campo con un intervalo de cinco minutos, y el segundo simula, de forma muy simple, el movimiento de cuatro jugadores dentro del campo de golf.

Ambos simuladores se ejecutan con el comando *node* seguido del archivo desde la carpeta *device_simulator*.