UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

DISEÑO E IMPLEMENTACIÓN DE UNA BASE DE DATOS Y UNA APLICACIÓN MÓVIL MULTIPLATAFORMA PARA EL CONTROL Y MONITOREO DE UNA EXPLOTACIÓN CUNÍCOLA

Pablo Vallés Juliá

Tutor: Ángel Héctor García Miquel

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2018-19

Valencia, 9 de septiembre de 2019

Escuela Técnica Superior de Ingeniería de Telecomunicación
Universitat Politècnica de València
Edificio 4D. Camino de Vera, s/n, 46022 Valencia
Tel. +34 96 387 71 90, ext. 77190
www.etsit.upv.es

VLC/ CAMPUS VALENCIA, INTERNATIONAL CAMPUS OF EXCELLENCE

EMAS Gestión ambiental verificada REG.NO. ES-CV- 000030

Resumen

En este proyecto se ha desarrollado una aplicación multiplataforma, realizada con el entorno de trabajo Ionic para dispositivos móviles, tanto para iOs como para Android, destinada a la domotización de una granja cunícula con el objetivo de monitorizar y controlar la cría de conejos, así como facilitar las tareas de la persona encargada de esta labor. También se ha desarrollado un pequeño servidor -Raspberry- de manera sencilla, en la cual se va a implementar una base de datos, donde se recogerá la información obtenida por diferentes sensores y cuyos datos se podrán visualizar y analizar en la aplicación desarrollada. En esta memoria también se explica cómo se ha llevado a cabo el diseño de la base de datos y de la API destinada a este proyecto, gracias a la cual podremos obtener esos datos mencionados anteriormente. Además, añade un apartado de futuras mejoras y posibles cambios para sacarle mayor rendimiento al proyecto, así como un pequeño pliego donde se recogen las condiciones para su puesta en marcha dentro de cualquier tipo de granja.

Abstract

This project explains the development of a multiplatform application, developed with the Ionic framework for mobile devices, both for iOS and for Android, aimed at the domotization of a rabbit farm in order to facilitate the tasks of the person who works there. It also explains how to mount a small server -Raspberry- in a simple way, in which a database will be implemented, where the information will be collected from different devices and whose data can be visualized and analyzed in the developed application. This report also explains how the database and API design for this project has been carried out, thanks to which we can obtain the aforementioned data. In addition, it adds a section of future improvements and possible changes to get more out of the project, as well as a bid specifications where the conditions for its implementation within any type of farm are collected.

Resum

En aquest projecte s'ha desenvolupat una aplicació multiplataforma, desenvolupada amb l'entorn de treball Ionic per a dispositius mòbils, tant per a iOs com per a Android, destinada a la domotització d'una granja de conills amb l'objectiu de monitoritzar i controlar la cria de conills, així com facilitar les tasques de la persona encarregada d'aquesta labor. A més, se explica com instal·lar un servidor senzill -Raspberry- en el qual es va implementar una base de dades, on es podrà arreplegar la informació obtinguda per diferents sensors per poder visualitzar i analitzar les dades en l'aplicació desenvolupada. En aquesta memòria també s'explica com es va portar a terme el diseny de la base de dades i de la API, destinada a aquest projecte, gràcies a la qual podrem obtenir aquestes dades mencionades anteriorment. A més, afexig un apartat de futures millores i possibles canvis per a aprofitar un major rendiment, així com un petit document on s'arrepleguen les condicions per a la seua posta en marxa dins de qualsevol tipus de granja.

Agradecimientos

En primer lugar, me gustaría agradecer, sin duda, a mi pareja por apoyarme y motivarme tanto con este proyecto, que sabe que es muy importante para mí. Por su sinceridad en cuanto a su opinión respecto al diseño de la aplicación y su visto bueno en cada una de las decisiones.

En segundo lugar, a mi tutor, Héctor García, por haber sido partícipe de este trabajo, supervisarlo y aconsejarme en todo momento que me veía perdido.

Por último, quería agradecer a mi familia, por soportar tanto estrés y ayudarme cuando lo necesitaba, sobre todo mi hermano pequeño.

Índice

Capítulo 1. Introducción

Los avances tecnológicos nos han acompañado desde 1769 cuando James Watt presentó la máquina de vapor, tiempo paralelo a la Revolución Industrial. A partir de este momento, se abrió una veda en tecnología que no conocería fronteras que la frenaran. Sumándose a este gran invento de Watt, otros hitos en tecnología fueron el sistema de producción en cadena traído de la mano de Henry Ford y las Tecnologías de la Comunicación y la Información (TIC) así como su causa: la globalización.

El siglo XXI se ha colgado "evolución" como concepto que le acompañará durante todos estos años que restan a este siglo. Ya se puede percibir que la mejora es constante en todos los ámbitos que nos rodean: coches casi autónomos, smart TV,'s teléfonos inteligentes, hogares domotizados, sistemas quirúrgicos robotizados como Da Vinci, etc.

La revolución 4.0 se ha adaptado a nuestras esferas privadas, profesionales y sociales para facilitarnos nuestra actividad en las tareas con avances como los siguientes: el Internet de las cosas (IoT), la inteligencia artificial, la comunicación M2M, el BigData, la fabricación digital e impresión 3D, etc.

Los avances son muchos y pese a que sea el sector terciario el que aviva la economía de un país, sigue siendo el sector primario el que lleva la ventaja con la materia prima, necesaria para nuestra supervivencia. Es por ello que es necesario adaptar las instalaciones de este sector a la revolución tecnológica que está viviendo nuestro siglo, para facilitar y agilizar los procesos de producción y recolección de alimentos.

Además de esto, los trabajos de este sector están siendo desempeñados principalmente por adultos en edades avanzadas, por lo que este tipo de aplicaciones pueden facilitar el trabajo a estas personas que cada vez se ven menos capacitadas para este tipo de trabajos físicos. Por otro lado, el acercamiento de este sector a la tecnología, visto siempre alejado de la modernidad y el avance, puede atraer a nuevos trabajadores jóvenes a las tareas relativas a la agricultura y ganadería dada su mayor comodidad, nada comparable con la del pasado, y su sinergia con la tecnología, que la hace más atractiva a los ojos del que trabaja.

Por todo esto, la revolución 4.0 ha de penetrar en el sector primario para cumplir una de sus funciones: acercar la comodidad a trabajos pesados con el objetivo de sacarle el mayor rendimiento posible. El desarrollo tecnológico ya está creado para todos los ámbitos, solo falta aplicarlo.

Capítulo 2. Motivación

Desde pequeño siempre me ha gustado la electrónica, la robótica y la programación, pero nunca he sabido por que camino quería decantarme seriamente. Hace 4 años, al entrar en la carrera de Ingeniería de Telecomunicaciones, me puse a juguetear un poco con la plataforma de Android Studio, e incluso comencé algún curso que nunca terminé por temas de tiempo y por no tener ningún proyecto en mente que me llenara de verdad.

Al escoger la rama de electrónica, poco a poco, me fui dando cuenta que lo que de verdad echaba de menos era sentarme delante de un ordenador con el objetivo de conseguir encender unas cuantas luces con unas líneas de código. Además, estaba todo el día con el teléfono móvil y las redes sociales, lo que hacía que mis ganas de desarrollar mi propia aplicación fueran aumentando de manera exagerada. Es ahí cuando un compañero de la carrera me propuso ayudar a su padre con la granja. La idea era domotizarla de tal forma que su padre pudiera ver el estado ambiental de la granja desde casa para que, en caso de que algo no funcionara correctamente, acudir lo antes posible y tratar de solucionarlo.

Pensé que sería una buena oportunidad para aprender de verdad a desarrollar aplicaciones móviles, pero, al estar tan cargado de tareas y no disponer casi de tiempo para poder desarrollar la idea, decidimos dejarlo de lado para ponernos con ello más adelante.

Al empezar el último curso de la carrera, mi compañero me volvió a proponer la idea y, con el Trabajo de Fin de Grado a la vista, pensé que no tendría otra oportunidad mejor que esta para obligarme a aprender lo que de verdad me motiva.

Al principio, la motivación era máxima. Conforme fue pasando el tiempo, nos enteramos de que una de las mayores empresas a nivel nacional relacionada con la carne de conejos estaba lanzando a nivel interno una aplicación para el control técnico de sus granjas. Por suerte, esta empresa era la misma para la que trabaja el padre de mi compañero que, además, conoce al subdirector de esta. Tras una reunión, tuvo la oportunidad de comentarle la idea de la aplicación del TFG y este mostró un gran interés en seguir adelante con el proyecto puesto que se trataba de un añadido muy importante para su aplicación. Fue entonces cuando la motivación de desarrollar este proyecto aumentó todavía más.

Actualmente, estoy desarrollando el propio TFG como un prototipo para presentarlo en la siguiente reunión que tendrá lugar a partir de septiembre puesto que esta empresa se encuentra en plena expansión y están actualmente instalando nuevas granjas en Cataluña.

Capítulo 3. Objetivos del proyecto

Tal y como se ha explicado en el apartado de "Introducción" de este trabajo, el objetivo principal de este proyecto es incorporar la tecnología en el sector primario, más concretamente en la ganadería, con el propósito de aumentar el rendimiento de esta, conseguir una considerable reducción de costes y un mejor aprovechamiento del uso de los piensos, que ha posibilitado la revolución industrial 4.0 además de una mayor eficiencia y comodidad para el granjero o granjera.

Otro de los objetivos perseguidos en este proyecto es encontrar un mayor bienestar para los animales con el fin de conseguir disminuir el número de defunciones y aumentar la calidad de la carne, así como los kilos de carne producidos.

Con este proyecto se pretende que se consiga una aplicación que informe al dueño o dueña de la granja sobre su estado, pero con el fin de llevarlo a que este lugar sea capaz de mantener unas condiciones óptimas para los animales de forma autónoma, así como informar al dueño del estado de *stock* o hacer pedidos en caso de que fuera necesario. Para ello, se ha utilizado una red de sensores para recopilar los siguientes parámetros: temperatura, humedad, cantidad de oxígeno o CO2 en el aire, la cantidad de pienso y agua y el estado de la calefacción y *cooling*. Esta serie de sensores se instalarán en varios puntos de la granja. Una vez obtenidas las medidas, se incorporarán automáticamente a una base de datos, a la que el usuario, mediante la aplicación desarrollada en este proyecto, podrá acceder para su consulta.

En este sentido, con la domotización de este tipo de granjas se pretende conseguir una atmósfera acorde a las necesidades de los animales con el fin de reducir el uso de los antibióticos en los conejos. De esta forma, se puede evitar la circulación masiva de medicamentos que afectan a la salud de los humanos. Cabe mencionar que este es uno de los puntos que se trató en el Parlamento Europeo y cuya decisión fue acabar con la puesta masiva de vacunas en animales para frenar el impacto medioambiental y sanitario que estas suponen. De hecho, desde el Gobierno se ha desarrollado una aplicación mediante la cual se pretende registrar los antibióticos utilizados en las ganaderías para controlar su consumo.

Además, con el desarrollo de esta aplicación se pretende llevar una contabilidad exacta del número de conejos existentes en todo momento en la granja, así como la cantidad de kilos que se han producido en cada una de las bandas para llevar un control más exhausto del lugar de trabajo. Esto facilita el trabajo tanto al ganadero como al veterinario sobre la información completa de la granja.

Con este proyecto también se persigue que el tiempo de respuesta ante cualquier problema que se pueda producir en la granja sea mayor, resolviendo de una forma más veloz y eficiente la incidencia. Además de esto, la domotización de estas grandes instalaciones que albergan cantidad de información en forma de kilos, número de conejos, litros etc. pueden desvelar necesidades y

consultas que puedan surgir en un futuro. De esta forma, la domotización de estos lugares anticipa al granjero o granjera la posibilidad de abordar estas cuestiones con antelación.

Capítulo 4. Metodología

Mi idea principal para este proyecto fue desarrollar una aplicación para el sistema operativo de Android, dado que durante la carrera había aprendido ciertos dotes sobre los lenguajes de programación que utilizaba dicho entorno de desarrollo. Comencé con el diseño de una interfaz principal con Android Studio.

De forma paralela a este proyecto se estaba desarrollando otro trabajo complementario, el cual consistía en instalar sensores en una granja para recopilar información. Estos datos se almacenan en la base de datos creada específicamente para el proyecto presentado en esta memoria con el fin guardarlos y trasladarlos a la aplicación desarrollada.

Durante mis prácticas curriculares en Everis, realicé un curso voluntario sobre Angular, un entorno de trabajo para el desarrollo de aplicaciones web. En dicho curso, también me informaron sobre la plataforma Ionic, un entorno de trabajo para desarrollar aplicaciones móviles y que condensa los sistemas operativos iOs y Android. Consideré más oportuno trabajar con esta nueva herramienta ya que ofrecía un amplio abanico de usuarios por trabajar con ambos sistemas. Además, Android Studio acababa de incorporar un nuevo lenguaje, Kotlin, del cual no había recibido formación alguna. Por lo tanto, el hecho de traspasar mi trabajo a la plataforma Ionic, desde mi punto de vista, fue un acierto ya que el curso recibido era reciente y ofrecía más posibilidades.

Dada la complejidad de este proyecto y la escasa formación obtenida de los lenguajes de programación empleados -JavaScript, HTML, CSS, PHP- así como del entorno de trabajo – Ionic, me ha sido necesaria la utilización de cursos académicos de varias páginas webs, como por ejemplo Udemy, para familiarizarme con todos estos recursos necesarios para mi trabajo de fin de grado.

Además, hice uso de la página GitHub, que hace uso del control de versiones Git. En dicha página subí el código para llevar un control de los cambios que iba realizando y, además, anotaba cambios y mejoras para su posterior desarrollo.

En primer lugar, desarrollé varios esquemas de varias posibles interfaces de usuario y se habló con el propietario de una granja cunícola situada en Utiel para ver cuál sería la forma más cómoda para el uso de la aplicación. A continuación, llevé a cabo el desarrolló de la interfaz que se pensaba que era la más adecuada, dejando de lado en un primer momento la conexión de la aplicación con la base de datos, ya que esta no estaba desarrollada todavía.

Una vez terminada la interfaz beta y obtenido el ordenador monoplaca, la Raspberry, que iba a funcionar como servidor, decidí componer un primer esquema de la base de datos. Para ello, me puse en contacto con la integradora Hermi. Esta empresa opera a nivel nacional y es la encargada

de la distribución de conejos de la mayoría de las granjas cunícolas de España. Dada la existencia de este grupo y el gran interés mostrado por este proyecto, consulté con ellos las necesidades hasta llegar a lo que fue el primer boceto de la base de datos.

En tercer lugar, desarrollé la API, a la cual la aplicación se conectaría para obtener los datos que fueran necesarios. Para este paso, me valí de cursos y tutoriales mediante los cuales aprendí los conocimientos suficientes para utilizar el lenguaje PHP. Para comprobar que todo estuviera correctamente desarrollado, hice uso del programa "Postman", con el cual pude verificar que las respuestas fueran las deseadas.

Por último, retoqué la interfaz e hice que la aplicación se conectara a cada una de las direcciones de la API, según la página o la acción que el usuario deseara hacer y comprobé que todo funcionara correctamente. En caso de que surgiera cualquier tipo de error, hice que la aplicación avisara de ello mediante un mensaje de error.

4.1 Diagrama temporal

En la siguiente figura se muestra un diagrama de Gantt, donde se puede observar la dinámica de trabajo seguida para llevar a cabo el proyecto. Tal y como se explica en el apartado anterior, comencé realizando una formación de Android Studio a través de cursos académicos online para desarrollar la aplicación. Al no tener conocimiento sobre ello, esta, junto con la formación de Ionic, fueron las tareas que más me ocuparon ya que tuve que adquirir las bases para sus posteriores desarrollos.

Como se observa en el diagrama, las tareas dedicadas al desarrollo de Android Studio (Formación y desarrollo de la primera interfaz) ocuparon un mes de mi trabajo, que, pese a no poder aprovechar todo este tiempo por cambiar de plataforma, me ayudó a mejorar la primera idea de interfaz y coger más confianza a la hora de programar.

Tras un mes de trabajo, decidí cambiar mi proyecto por completo con el entorno de trabajo Ionic. Este cambio supuso perder las horas dedicadas en la plataforma anterior, pero ganar más terreno gracias a poder desarrollar una aplicación multiplataforma. Como se ha comentado anteriormente, la formación Ionic fue una de las tareas que más tiempo me llevó, ya que fue en total 29 horas de curso y más de 50 artículos de material extra.

Durante el transcurso del aprendizaje de Ionic, fui enfocando lo aprendido al diseño de la interfaz. Además de esto, al haber desarrollado un primer diseño para Android, la dificultad fue menor, por lo que agilicé el proceso y, en vez de 10 días que me duró crear este primer boceto de la aplicación, me llevo tan solo 7 días.

Respecto a la configuración de la Raspberry, me encontré con dificultades para abrir los puertos del router y poder acceder a esta, tanto dentro como fuera de la red, e incluso llegué a asignarle

una dirección fuera del rango de direcciones asignables por mi router, por lo que tuve que repetir el proceso de toda la configuración un par de veces. Como norma general, podría estar lista en unas cuantas horas, pero los obstáculos ralentizaron mi actividad.

En cuanto a la configuración del servidor, me costó más de lo normal debido a que la información que encontré para llevarla a cabo no estaba disponible en los repositorios que se indicaban. Por ello, estuve varios días hasta encontrar los comandos, gracias a los cuales conseguí instalar tanto Apache como PHP.

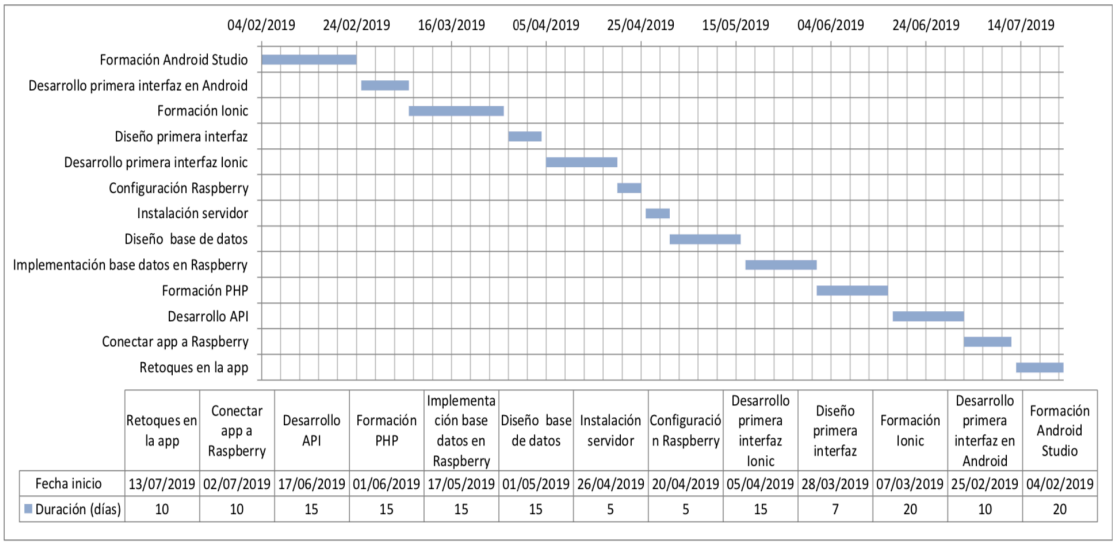
En cuanto al diseño de la base de datos, en primer lugar, la llevé sobre papel. Busqué un esquema generalizado (figura 23) para su diseño con la finalidad de que este pudiera ser utilizado por cualquier granja, incluso de otro tipo que no fuera cunícula. Además, traté de buscar un modo en el que todo estuviera relacionado para que, de una forma más cómoda y sencilla, poder añadir una granja o, incluso, nuevos sensores.

Al implementar la base de datos, fui viendo cambios que se podían adaptar mejor para un diseño más sencillo e incluso descubrí información que no había tenido en cuenta en el primer diseño. La gran cantidad de información que se pretende guardar en estas bases de datos hizo que esta implementación me llevara más tiempo de lo normal.

Para el desarrollo de la API, hizo falta también una pequeña formación, pero esta vez en lenguaje PHP. Para ello me valí de varios cursos y tutoriales, gracias a los cuales pude llevar a cabo el desarrollo necesario de mi siguiente paso.

Respecto a la conexión de la aplicación con la base de datos, no era costosa, sino más bien repetitiva. El problema que se presentó fue que la API desarrollada, en algunos casos, no llegaba a devolver la información ordenada de la forma en la que se requería. Por ello, la conexión de la base de datos con la aplicación se convirtió también en unas pequeñas pinceladas al desarrollo de la API.

Por último, al probar mi proyecto, me di cuenta de que había pequeñas mejoras como no poder retroceder a una etapa anterior a la hora de crear una nueva banda o que no se guardaban correctamente las defunciones. Además, aproveché para darle un toque más estético y atractivo para el usuario, así como crear un logo propio que pusiera el broche final a esta primera etapa del proyecto.



	Retoques en la app	Conectar app a Raspberry	Desarrollo API	Formación PHP	Implementación base datos en Raspberry	Diseño base de datos	Instalación servidor	Configuración Raspberry	Desarrollo primera interfaz Ionic	Diseño primera interfaz	Formación Ionic	Desarrollo primera interfaz en Android	Formación Android Studio
Fecha inicio	13/07/2019	02/07/2019	17/06/2019	01/06/2019	17/05/2019	01/05/2019	26/04/2019	20/04/2019	05/04/2019	28/03/2019	07/03/2019	25/02/2019	04/02/2019
Duración (días)	10	10	15	15	15	15	5	5	15	7	20	10	20

Fig. 1 Diagrama de Gantt

Capítulo 5. Desarrollo del proyecto

A continuación, se detallarán los diferentes puntos a desarrollar para poder llevar a cabo el proyecto.

5.1 Instalación en la granja

En primer lugar, se deben instalar los sensores pertinentes. Para el desarrollo de este proyecto, se han utilizado los siguientes sensores: temperatura, humedad, cantidad de oxígeno y CO2 en el aire, ultrasonidos para estado del agua (litros) y del pienso (kilos) y el estado de la calefacción y refrigeración (*cooling*). La instalación de estos sensores pertenece al proyecto comentado en el apartado de la metodología que se desarrolla de forma paralela a este trabajo.

Para los sensores de medidas medioambientales, se instalarán tres pequeñas bases a lo largo de cada túnel. Cada una de ellas medirá la temperatura, la humedad, la cantidad de oxígeno y la cantidad de CO2. Estas bases mandarán sus respectivas medidas a una *Raspberry* instalada en la propia granja. Este pequeño ordenador calculará la media de los valores recibidos por cada una de las bases. El resto de los sensores (tanque de agua, pienso y calefacción) también mandarán sus lecturas a este pequeño servidor. Cada media hora, la *Raspberry* subirá a una base de datos albergada en un servidor externo todos los datos recopilados.

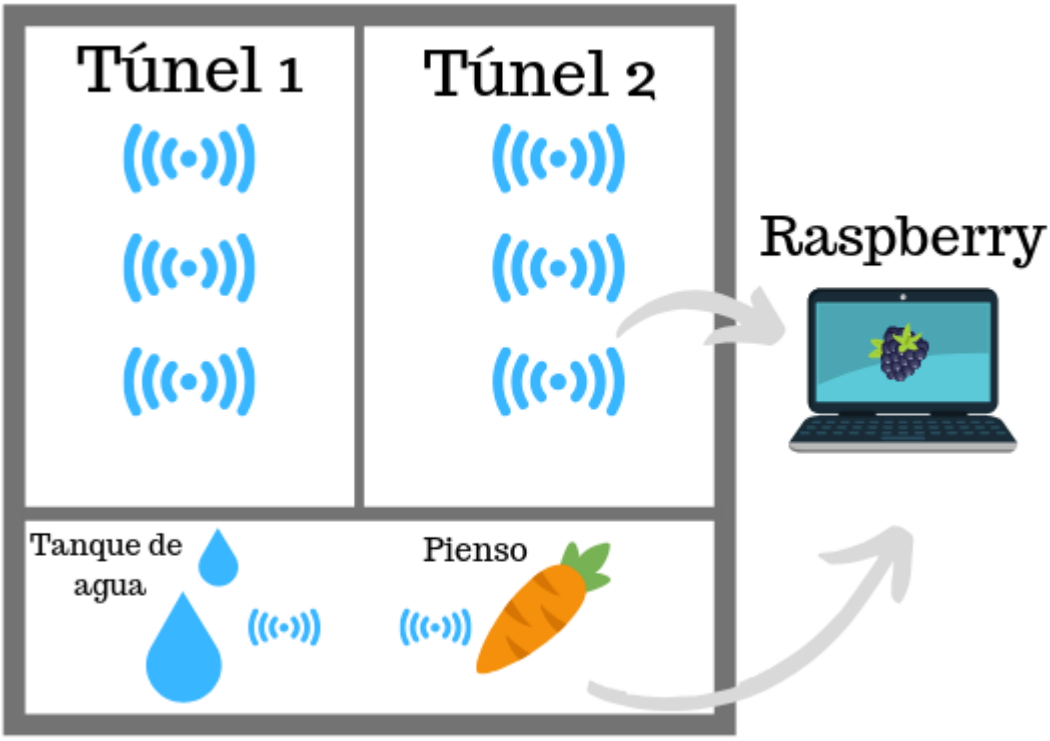


Fig. 2 Esquema de instalación dentro de la granja

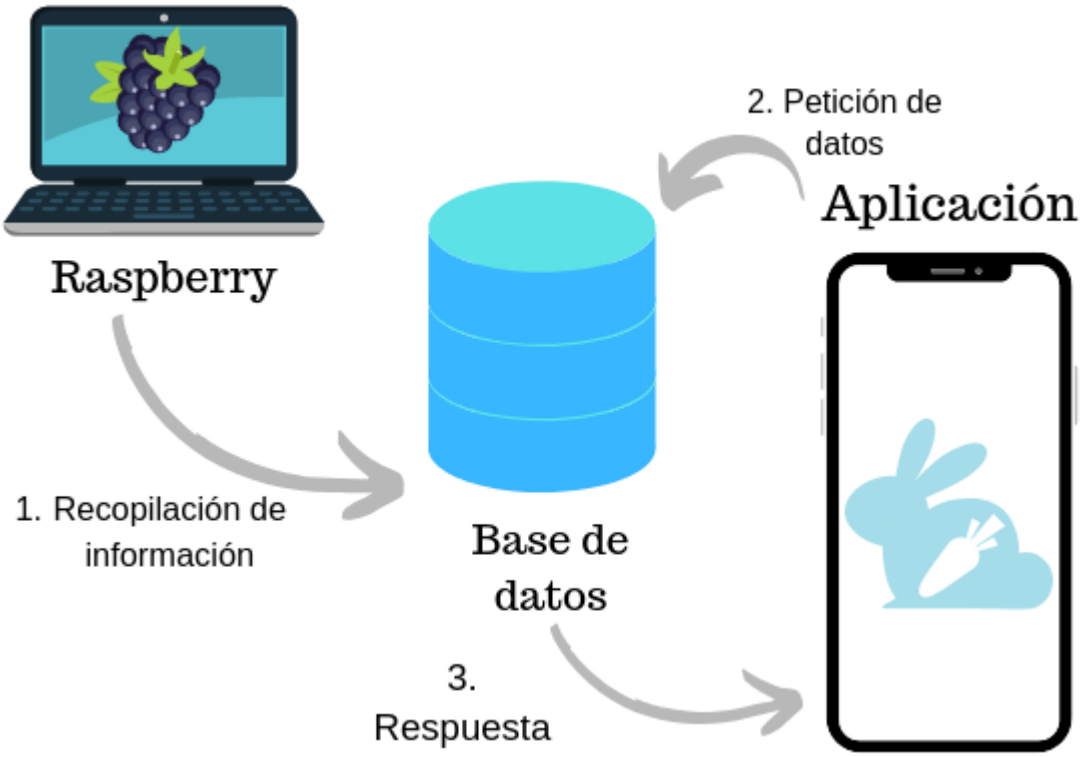


Fig. 3 Flujograma de datos

5.2 Aplicación móvil

5.2.1 Aplicaciones nativas y aplicaciones híbridas

Cabe diferenciar entre varios tipos de aplicaciones antes de comenzar con el desarrollo: aplicaciones nativas y aplicaciones híbridas.

Las aplicaciones nativas, son aquellas que están desarrolladas en un lenguaje especifico según el sistema operativo para el que estén siendo desarrolladas: *Java* o *Kotlin* para Android y *Objective-C* o *Swift* para Apple. La principal ventaja de este tipo de apps es la velocidad de ejecución y el rendimiento que se obtiene de la aplicación. Además, con este tipo de aplicaciones, se puede hacer uso de cualquiera de las funcionalidades que ofrecen los dispositivos móviles como por ejemplo la cámara, el micrófono, etc. Por otro lado, estas apps deben desarrollarse una vez por cada plataforma en la que se desee tener opción a instalar provocando así un mayor coste tanto en lo económico como en el tiempo de desarrollo.

En cambio, las aplicaciones híbridas, son aquellas que se desarrollan mediante el uso de la tecnología web como *HTML*, *CSS* y *JavaScript*. Estas apps están más limitadas en cuanto al uso de las funcionalidades del teléfono, aunque cada vez son menos los elementos que no se pueden incorporar a este tipo de apps. Aunque es cierto que el rendimiento es ligeramente inferior que el

de las aplicaciones nativas, la capacidad de poder portar un único desarrollo a los diferentes sistemas operativos que existen hoy en día en el mercado hace que el desarrollo de aplicaciones híbridas aumente cada día de manera considerable.

5.2.2 Ionic

Ionic es una herramienta o entorno de trabajo pensado para la creación de aplicaciones híbridas de una manera rápida y sencilla. Con Ionic se puede desarrollar aplicaciones tanto para móviles como para tablets de manera sencilla utilizando los lenguajes *HTML*, *CSS* y *JavaScript*.

Este framework esta basado en *AngularJS*, un entorno de trabajo de código abierto utilizado para el desarrollo de páginas web y que es el encargado de controlar y crear las interacciones dentro de la aplicación.

Para poder empaquetar el código y compilarlo a una aplicación nativa será necesario el uso del framework de Apache Cordova mediante el cual también se podrá acceder a funcionalidades del teléfono.

SASS y *GulpJS* son librerías mediante las cuales Ionic aportará estilos a la aplicación.

Por último, también será necesario tener instalado *NodeJS* para la instalación de Ionic, la creación de proyectos y el servicio de poder previsualizar la app en un navegador web sin necesidad de empaquetar ni compilar.

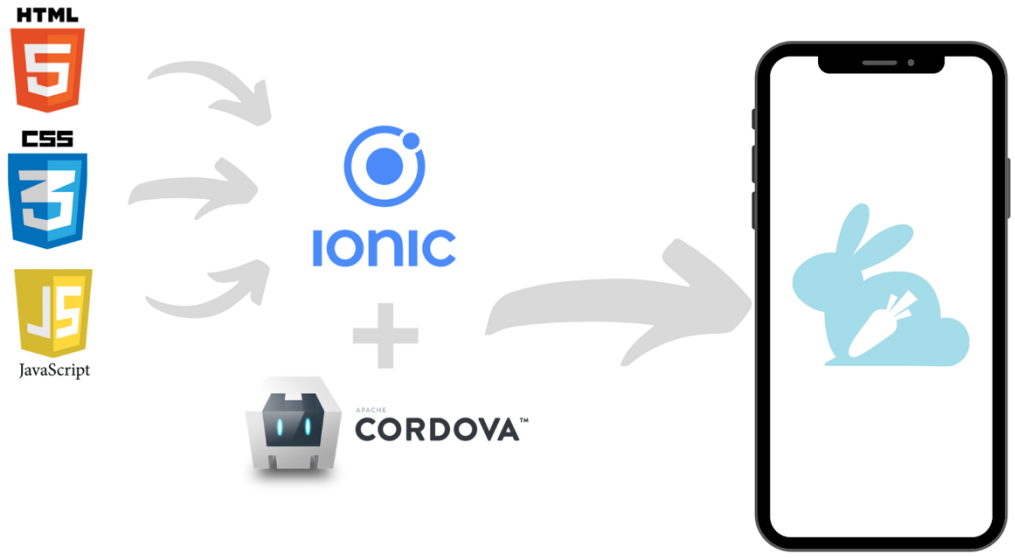


Fig. 4 Componentes del entorno de trabajo Ionic para aplicaciones multiplataforma

5.2.3 Desarrollo de la aplicación del proyecto

Para desarrollar la aplicación, se ha planteado, principalmente, un punto básico en cualquier aplicación destinada al uso por parte de diferentes usuarios: la pantalla de inicio de sesión. En esta pantalla se podrá distinguir tres elementos: el logo de la aplicación, dos campos de texto en los que el usuario deberá introducir sus credenciales y un botón mediante el que se conectará con la base de datos para comprobar que esas credenciales sean las correctas. En caso de que las credenciales introducidas no sean las correctas, se mostrará un mensaje en la parte baja de la pantalla para avisar al usuario.

Una vez verificadas las credenciales, la aplicación se conecta automáticamente con la base de datos para recibir un listado con las diferentes granjas que pertenecen a dicho usuario. Estas granjas se mostrarán en la siguiente página de la aplicación en forma de lista para seleccionar aquella de la cual se quiere obtener la información. Una vez seleccionada, la aplicación nos redirigirá al menú principal.

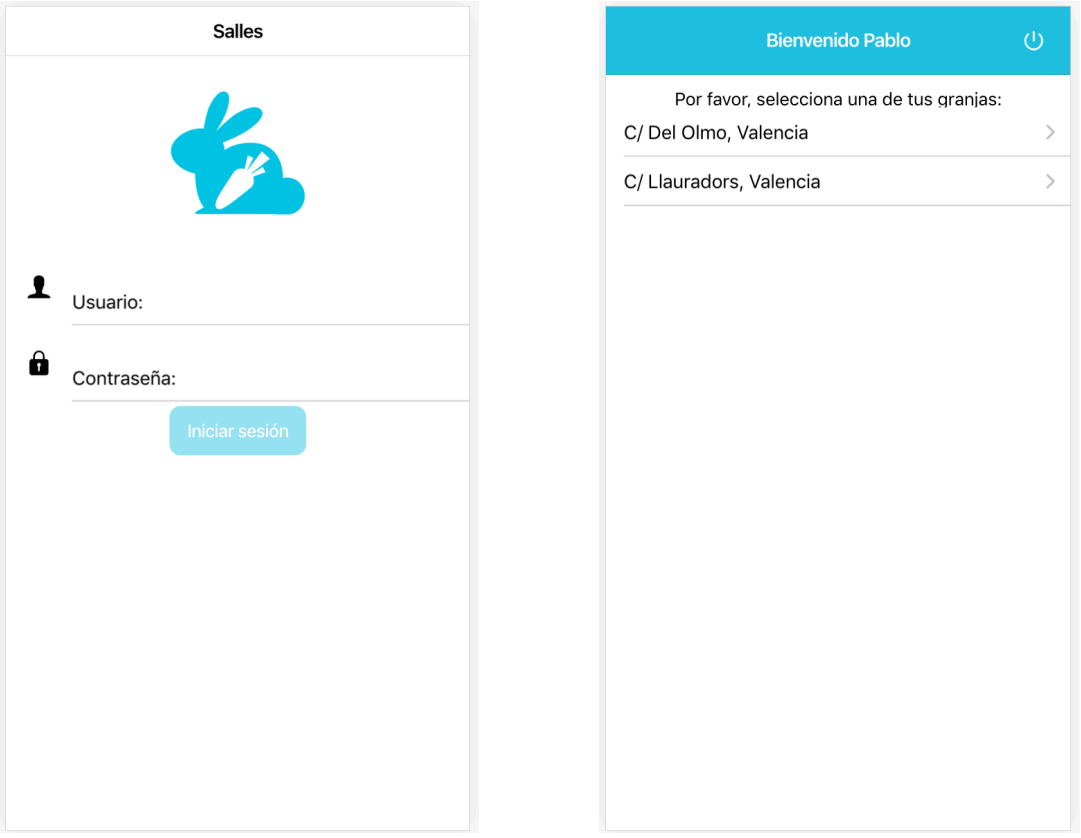


Fig. 5 Capturas de pantalla de la página *login* (izq.) y selección de granjas (drcha.)

En el menú principal se mostrará una lista con diferentes apartados entre los que se podrá distinguir: el estado general de la granja, el estado de los silos de pienso y agua, la climatización

de la granja, un apartado donde poder visualizar gráficas y otro donde registrar todos los datos técnicos.

En el "estado general" se cargará el número de túneles que tiene dicha granja de tal forma que se creará una carta por cada túnel y dentro de esta se indicará los valores de las últimas lecturas de los sensores de temperatura, humedad, oxígeno y CO2 guardadas en base de datos. Al final de línea de cada medida se mostrará un botón que dirigirá al usuario a la página de gráficas dejando seleccionado el sensor que se pulse para mayor comodidad por parte del usuario a la hora de crear las gráficas.

Cada medida dispone de un botón a final de línea que redirigirá al usuario al apartado gráficas. Este vínculo selecciona automáticamente el sensor escogido en el botón del "Estado General" (ej. temperatura, humedad, CO2, oxígeno)

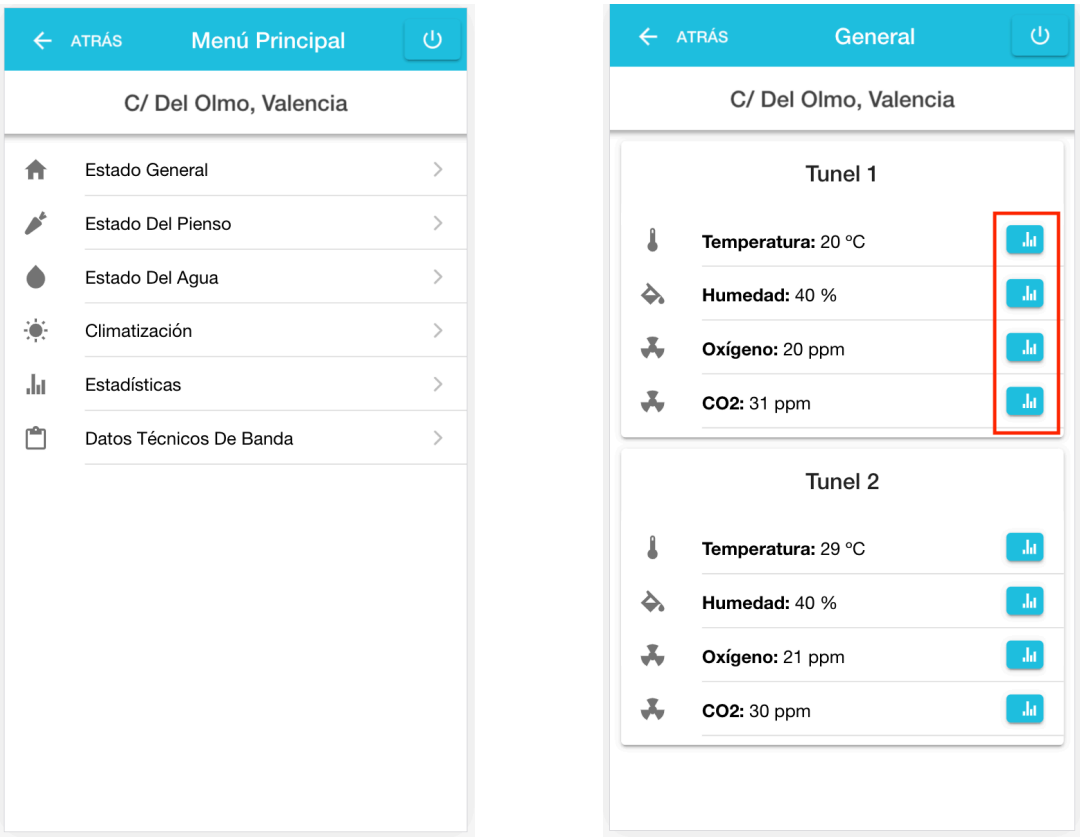


Fig. 6 Capturas de pantalla del menú principal (izq.) y del estado general (Drcha.)

En todas las granjas se puede diferenciar entre varios tipos de piensos. En las granjas cunícolas, se puede diferenciar entre pienso de engorde, pienso maternal y pienso de retirada. En el apartado de "estado del pienso" se mostrará la cantidad de kg disponibles en los silos de cada uno de los

diferentes tipos de piensos mencionados anteriormente. En el apartado "estado del agua" también se mostrará la cantidad de agua disponible en el tanque de agua.

La página "climatización" nos indicará si el aire acondicionado, la calefacción y los mecanismos de ventilación están encendidos o apagados. Los botones que aparecen en esta pantalla solo son indicadores del estado de estos mecanismos, sin posibilidad de controlarlos mediante la aplicación.

Una diferencia importante dentro de este proyecto es la posibilidad de crear gráficas con los datos almacenados en la base de datos con una resolución diaria o mensual. En la pantalla "gráficas" se podrá seleccionar una fecha de inicio y una fecha final para indicar la ventana de tiempo con la que se desea escoger los valores a representar. Se debe seleccionar también el sensor o los diferentes datos que se desean graficar y el tipo de gráfica que se desea: de líneas o de barras.

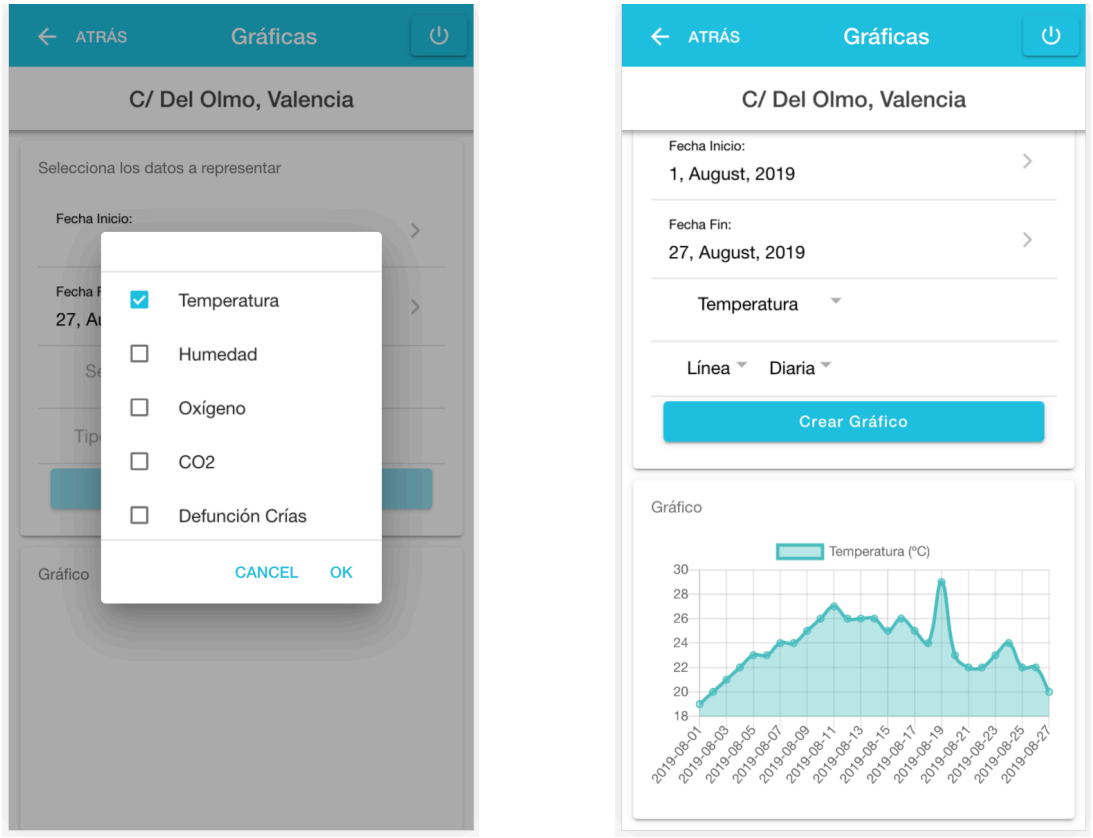


Fig. 7 Capturas de pantalla de gráficas de temperatura

Para conseguir trazar las gráficas, se han de recoger las fechas una por una, por lo que se ha tenido que hacer una serie de bucles, los cuales vayan rellenando un vector con cada una de las fechas, desde la fecha de inicio hasta la fecha de fin. En caso ser resolución mensual, se han tenido en cuenta únicamente los meses y el año y, en caso de ser resolución diaria, se ha ido aumentando el número de día de uno en uno hasta llegar al último día de cada mes (28 en caso del mes de febrero).

Una vez se ha llegado a dicho día, se reestablece el día 1 y se aumenta al mes al siguiente hasta llegar a la fecha seleccionada como fecha fin.

Para poder dibujar las gráficas, se ha hecho uso de la librería CHART, la cual nos brinda un gran abanico de posibilidades entre las cuales se han escogido las gráficas de líneas (Figura 2) y de barras (Figura 3 izq.). Además, esta librería nos proporciona la opción de, entre tanto dato dibujado en las gráficas, seleccionar uno de los días o meses representados y obtener el valor concreto.

En caso de elegir varios sensores para trazar la gráfica, se ha hecho que se muestre todo sobre la misma para así poder comparar y sacar conclusiones.

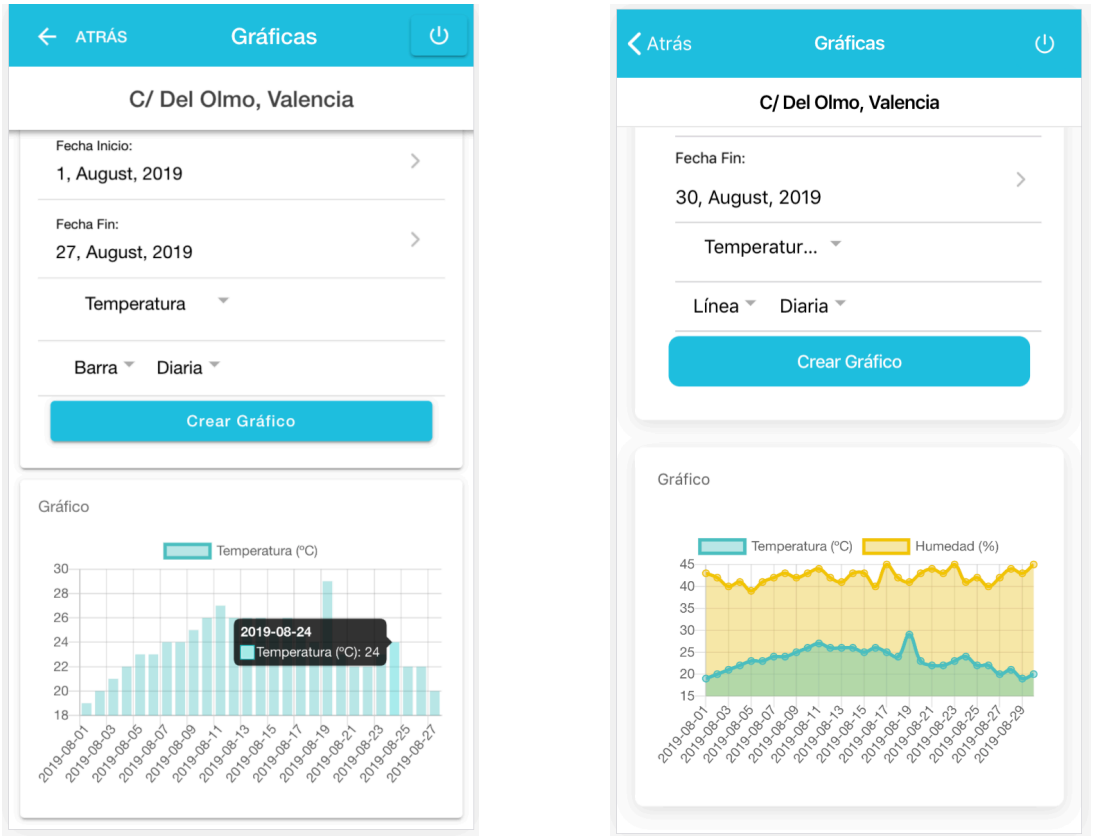


Fig. 8 Captura de pantalla de representación de barras (izq.) y comparación entre varios sensores (drcha.)

Las granjas cunícolas se organizan en bandas. Una banda es el conjunto de conejas que tienen la misma fecha de parto.

Para iniciar una banda, se deben realizar lo que se denomina cubriciones. Las cubriciones son los procesos de inseminación a las conejas que se han preparado para la misma banda. Si se quiere evitar tener partos en fines de semana, las cubriciones deberán hacerse únicamente los lunes y los

viernes. El número de conejas inseminadas en cada una de las bandas dependerá de las jaulas que se tengan preparadas para este proceso de la cría.

La siguiente fase sería la de las palpaciones. Esta se realizará 10 días después de las cubriciones. Si se ha tratado de evitar partos en el fin de semana, las palpaciones siempre caerán en lunes o jueves. En este paso se comprobará que conejas están en proceso de gestación.

A los 28 días desde el inicio de la banda, las conejas que continúen en proceso de gestación se cambiarán a unas jaulas diferentes denominadas nidales para que, 3-4 días más tarde, paran a sus gazapos.

Entre los 25 -28 días después del parto se retiran los nidos de las jaulas y, 5-7 días después tiene lugar el destete. En dicha fase, los conejos son separados de las madres y se llevan a lo que se conoce como jaulas de cebo donde se juntará con sus hermanos para criarlos de forma conjunta.

Además, 7-8 días después del parto, se vuelve a realizar las cubriciones con las conejas de dicha banda para que, 3-4 días después del destete, estas conejas estén pariendo de nuevo.

La página “datos técnicos de banda” mostrará una lista con todas las bandas registradas y relacionadas con la granja del usuario. En esta lista se muestra la fecha de inicio de la banda, los días que han pasado desde que se inició y el estado en el que se encuentra. En este caso, el servidor nos devuelve las fechas con formato año/mes/día, y la aplicación las ordena en día/mes/año, para una mayor comodidad y comprensión para el usuario. Este podrá acceder a la información de cada una de ellas con el simple hecho de pulsar sobre el elemento de la lista. Además, también se puede ver un botón mediante el cual se podrán crean nuevas bandas.

Si el usuario selecciona una de las bandas del listado y si estado es el número 1, la aplicación redirigirá al usuario a la página “Palpaciones”, si el estado es el 2, se redirigirá a la página “Actualizar Banda” y en caso de que la banda se encuentre en estado finalizado, se redirigirá a la página “Resumen de banda”.

Para crear una banda se debe indicar la fecha de inicio y el número total de madres que han sido inseminadas. Para indicar la fecha de inicio, la aplicación selecciona automáticamente la fecha actual, pero ofreciendo la posibilidad al usuario de modificarla. Será entonces cuando, pasaremos al estado 1 de la banda: las palpaciones.

En la pantalla de palpaciones se podrá guardar el número de palpaciones y el número de nacimiento de crías a través del botón “guardar palpaciones”. Este botón se activará únicamente si alguno de estos dos campos contiene caracteres numéricos. En caso de introducir cualquier tipo de carácter no numérico, el botón se desactivará. También se dispone de acceso a las páginas “defunciones madres” y “defunciones crías” donde se indicará la fecha de registro de la defunción y en cada uno de los motivos se anotará el número de conejos que han fallecido por dicho motivo.

Por último, en la página “palpaciones”, habrá un acceso directo a la pantalla “resumen de banda” donde se indicará, de manera clara, todos los datos guardados de dicha banda.

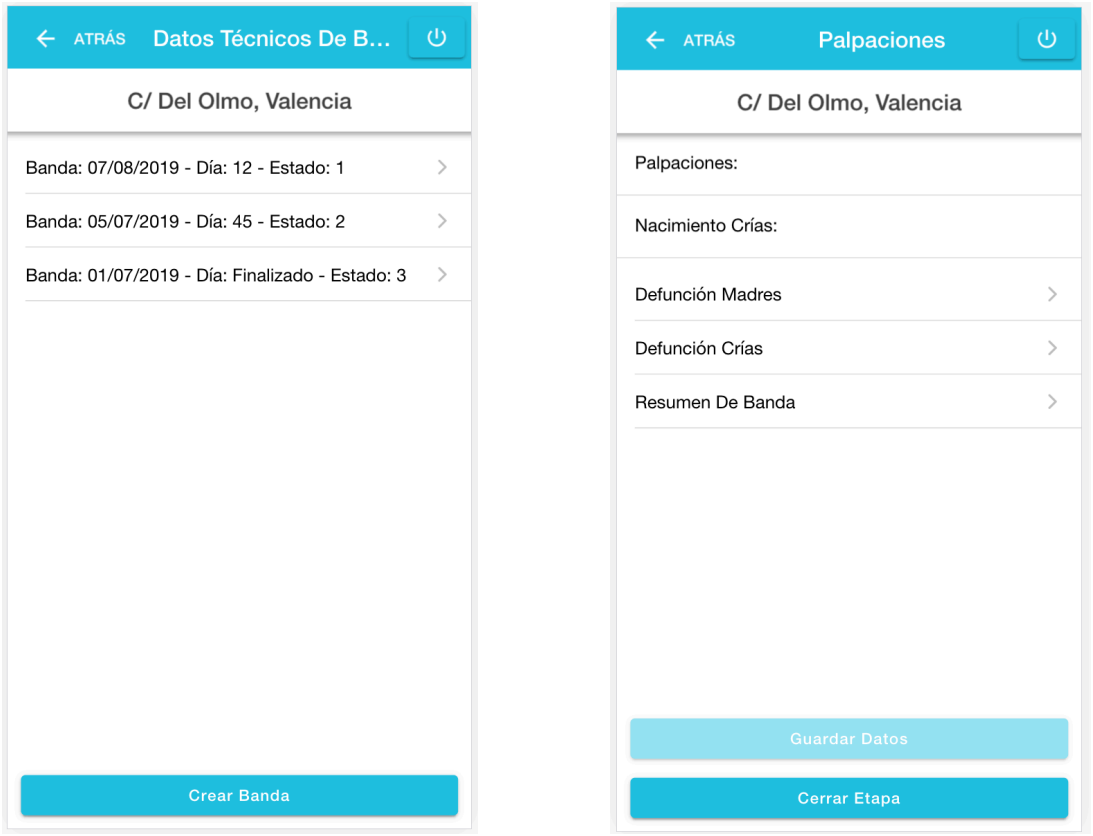


Fig. 9 Capturas de pantalla del menú datos técnicos de banda (izq.) y la ventana de palpaciones (drcha.)

Una vez cerrada la etapa de palpaciones, la banda pasará a estar en el estado 2, el estado “actualizar banda”. En esta pantalla solo se podrá acceder a las pantallas “defunción gazapos” y “resumen de bandas” ya que en esta etapa de la banda las madres ya habrán sido separadas de las crías y estarán siendo preparadas para empezar otra banda.

Tras finalizar esta etapa, la banda pasará al último de los estados, el estado 3. En este estado se mostrará la pantalla “resumen de banda” con todos los datos que se hayan guardado.

5.3 Raspberry

Para el desarrollo de este prototipo se ha utilizado una *Raspberry Pi 3* a modo de servidor en el que se guarda la base de datos que contiene toda la información sobre los usuarios, mediciones y control de los datos técnicos.

El bajo coste de este ordenador de placa reducida y su bajo consumo convierten a la *Raspberry* en el servidor ideal para el desarrollo de prototipos.

Este ordenador no incluye ningún tipo de periférico, aunque cuenta con puertos USB, HDMI, Ethernet y una ranura para tarjeta MicroSD, así como memoria RAM, GPU, 40 pines GPIO y un procesador Broadcom.

5.3.1 Configuración de la Raspberry

La *Raspberry* viene sin sistema operativo instalado por defecto, por lo que, se necesita una tarjeta MicroSD donde instalarlo. En este caso se ha escogido Raspbian, una distribución de Linux basada en Debian y orientada a la enseñanza de informática, ya que es el sistema operativo proporcionado de forma oficial por la propia distribuidora *Raspberry Pi Foundation* y reconocido como el principal sistema operativo para estos ordenadores.

Este *software* libre y de código abierto se puede obtener a través de la página oficial de www.raspberrypi.org donde se puede encontrar en 3 versiones diferentes, 2 de estas cuentan con escritorio instalado por defecto mientras que la versión Lite, la mínima imagen de Raspbian, no lo trae instalado.

Debido a que una vez terminada toda la instalación y configuración necesaria no se requiere ningún uso de interfaz gráfica por parte del usuario, la versión descargada e instalada será la versión Raspbian Lite aprovechando, además, que el tamaño de esta imagen es menos de la mitad que el tamaño de cualquiera de las otras dos versiones.

Para extraer los archivos de la imagen del Zip descargado, la propia fundación aconseja hacer uso del programa *7Zip* en Windows y *The Unarchiver* en macOs ya que se tratan de programas totalmente gratuitos y han sido probados por los propios desarrolladores para extraer los ficheros correctamente.

Una vez descargada la imagen, se precisa un programa con el cual se grabará la imagen a la tarjeta MicroSD, como por ejemplo *Win32DiskImager* para Windows o *ApplePi-Baker* para macOS. Estos programas son muy sencillos de utilizar, pero se debe tener en cuenta que, antes de grabar la imagen, se debe formatear la tarjeta y, por lo tanto, se perderán todos los archivos guardados.

Se llevará a cabo lo que se conoce como una configuración Headless ya que ni en la instalación ni durante su uso se utilizará ningún tipo de periférico como monitor, teclado o ratón. Para ello será necesario un ordenador remoto con el que poder acceder al servidor mediante el uso de protocolos SSH y SFTP. A continuación, se describen ambos protocolos:

SSH (Secure Shell)

SSH es un protocolo de red mediante el cual los usuarios pueden controlar y modificar de forma remota sus servidores a través de un canal seguro en el que toda la información está cifrada.

SFTP (Secure File Transfer Protocol)

SFTP es un protocolo basado en SSH para la transmisión de archivos de forma segura entre tu ordenador local y el servidor remoto. Además de posibilitar la opción de transferir ficheros, permite visualizar de forma clara y sencilla los directorios del servidor, así como cambiar el nombre o limitar los derechos de los usuarios sobre ficheros y directorios.

Antes de iniciar la *Raspberry* por primera vez, se habilitará la opción SSH que por defecto y como medida de seguridad viene desactivada. Para poder dejar la opción SSH activada antes de arrancar el ordenador se debe crear un archivo de texto dentro de la tarjeta MicroSD con el nombre de *ssh.txt*. Esto activará automáticamente la opción SSH al encender el servidor.

Por último, antes de arrancar la *Raspberry*, se debe asegurar conexión a la red de internet que, aunque es muy aconsejable que esta sea mediante conexión por cable Ethernet debido a la seguridad, la velocidad y la estabilidad, también se puede mediante conexión wifi. Para esta última, se debe crear otro fichero llamado *wpa_supplicant.conf* que también se debe añadir a la tarjeta MicroSD junto con el fichero *ssh.txt* y que tendrá el siguiente aspecto y donde sustituiremos el nombre y la contraseña de la red a la que se desea conectar dentro de las comillas:

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
country=ES

network={
    ssid="nombre de la red wifi"
    psk="contraseña de la red wifi"
    key_mgmt=WPA-PSK
}

```

Fig. 10 Configuración del archivo wpa_supplicant.conf

Una vez arrancado el servidor, se necesita saber la dirección IP dinámica asignada por el *router* a la *Raspberry* dentro de nuestra red local. Para ello, se puede hacer uso de programas de escáner de IPs como *Advanced IP Scanner* para Windows o *IP Scanner* para macOS. Si se cuenta con acceso a la configuración del *router* al que esta conectado nuestro servidor, se puede conocer la dirección IP accediendo al mapa de red donde también aparecerán las direcciones IP de cada uno de los dispositivos conectados a la red.

Para poder acceder al servidor desde Windows, es necesario disponer de un cliente de SSH como *Putty*. Desde macOS o Linux será suficiente con abrir la consola de comandos y escribir "ssh nombreDeUsuario@direcionIP". En este caso, al instalar la imagen de Raspbian, el nombre de usuario por defecto es "pi" y la contraseña "raspberry", mientras que, como dirección IP, se escribirá la dirección IP dinámica asignada por el *router* a la hora de conectarse el servidor a la red y que previamente se ha obtenido desde el mapa de red o bien con cualquiera de los programas de *scanner* de IPs.

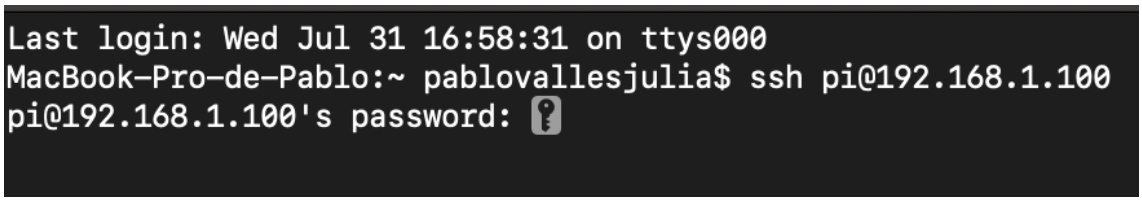


Fig. 11 Ejemplo de conexión ssh desde macOs

Una vez se pueda acceder al servidor, se comenzará con la configuración de la *Raspberry*. En primer lugar, mediante el comando "sudo raspi-config" se accede a la utilidad de configuración y, mediante las opciones avanzadas, se le dará permiso a la *Raspberry* para que haga uso de todo el espacio disponible en la tarjeta MicroSD mediante la opción "Expand Filesystem".

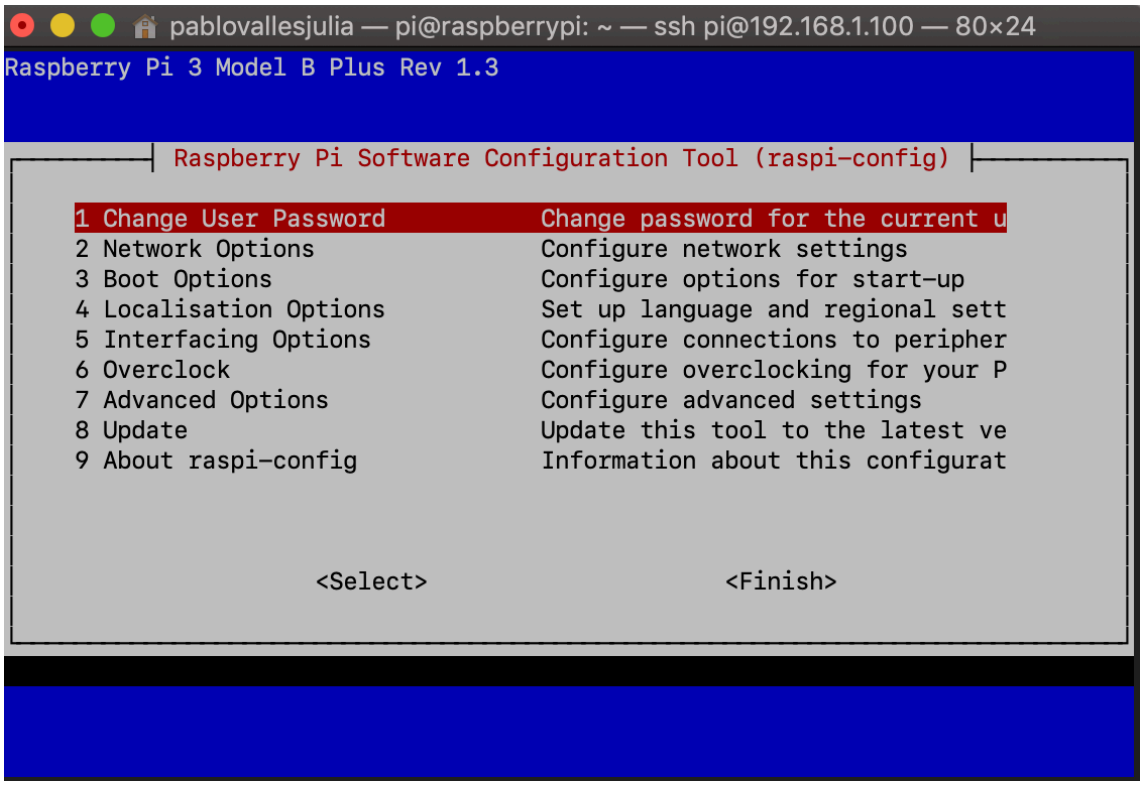


Fig. 12 Captura de pantalla de la utilidad de configuración

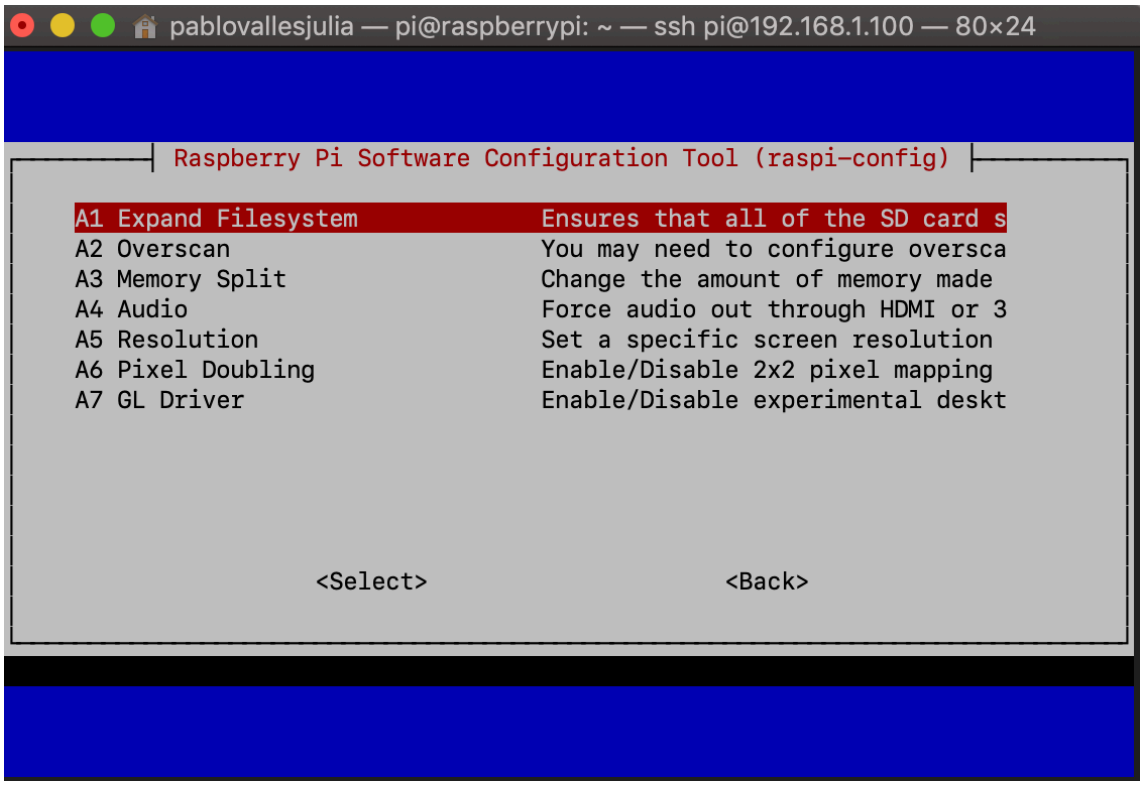


Fig. 13 Opción de expandir el uso de la memoria

A continuación, fuera de las utilidades de configuración, se hará uso de los comandos "sudo apt-get update" para actualizar la lista de paquetes con las últimas versiones disponibles en los repositorios de Raspbian y "sudo apt-get upgrade" para actualizar los paquetes instalados a las últimas versiones disponibles.

La dirección IP de la *Raspberry* es asignada por el *router* mediante el protocolo DHCP, un protocolo de red mediante el cual el *router* asigna una dirección IP dentro de un rango de direcciones, así como otros parámetros de configuración de red a cada uno de los dispositivos que se conecten a dicha red. Estas direcciones se van asignando según van quedando libres conociendo en todo momento quién ha estado en posesión de esa dirección, por cuánto tiempo y a quién se le asignó después.

Puedes ver las direcciones IP dinámicas asignadas por el servidor DHCP

dirección IP dinámica

nombre	dirección IP	dirección MAC
android-21f3b25e440beee2	192.168.1.42	4C:74:03:3D:EB:69
LAPTOP-E5JHAVKK	192.168.1.44	A0:D3:7A:E7:D2:13
iPhonedlesJulia	192.168.1.49	48:43:7C:31:37:EB
iPhone-de-Pablo	192.168.1.54	D0:C5:F3:BA:61:08
Unknown Device	192.168.1.57	D4:4B:5E:C3:C3:46
Raspberry	192.168.1.100	B8:27:EB:AE:AD:10
Dani	192.168.1.117	A0:D7:95:E6:9E:DA
MBP-de-Pablo	192.168.1.129	AC:BC:32:BB:BB:DB

Fig. 14 Lista de direcciones IP asignadas por el router

Para el servidor será necesario asignar una dirección IP fija a la *Raspberry* para que dicho protocolo no la modifique. Para ello, se debe modificar el fichero *dhcpcd.conf* mediante el comando *"sudo nano /etc/dhcpcd.conf"* de tal forma que tenga el siguiente aspecto:

```
# Example static IP configuration:
interface eth0
static ip_address=192.168.1.100/24
static routers=192.168.1.1
static domain_name_servers=192.168.1.1
```

Fig. 15 Configuración de dirección IP estática

En la línea de *interface* se indicará el nombre de la interfaz que se quiere configurar utilizando "eth0" para conexión por cable *ethernet* y "wlan0" para conexión por wifi. En *static ip_address*, se escribirá la dirección IP estática que se desee siempre y cuando esté fuera del rango de direcciones asignables por el DHCP del *router*. En *static routers* y *static domain_server_servers* se escribirá la dirección del *router* que por defecto suele ser la 192.168.1.1.

Por último, se reiniciará la *Raspberry* con el comando "sudo reboot" para aplicar todos los cambios realizados.

5.4 Configuración del servidor

Para comenzar con la configuración del servidor será necesario 2 cosas básicas y fundamentales, Apache y PHP.

APACHE2

Apache2 se trata de un *software* de servidor web de código abierto y totalmente gratuito. Este *software* ligero, fiable y flexible, es el encargado de mostrar el contenido en más del 50% de las páginas web en las que los usuarios navegan diariamente, así como proporcionar al dueño de la web la capacidad de gestionar de forma sencilla los ficheros de su página.

PHP

PHP se trata de un lenguaje de programación de código abierto y utilizado para el desarrollo web del lado del servidor y cuya característica principal es la posibilidad de ser implementado en archivos HTML generando así páginas web dinámicas. Por defecto, Raspbian dispone de PHP versión 5, sin embargo, su última versión 7 es mucho más rápida y eficiente.

MYSQL

MySQL es el motor de base de datos relacional más conocido del mundo. Se trata de un software de código abierto distribuido por Oracle Corporation cuya función es tanto almacenar como administrar los datos que se almacenan en las bases de datos.

Fig. 16 Software necesario para la configuración del servidor

La instalación de Apache se lleva a cabo mediante el comando "sudo apt-get install apache2" mientras que para la instalación de PHP usaremos el comando "sudo apt-get install php". Mediante estos comandos se obtienen directamente las últimas versiones de todos los paquetes de Apache2 y PHP.

Para comprobar que todo funciona correctamente, se puede crear un archivo de prueba en la ruta /var/www/html con el nombre “infophp.php” y cuyo contenido sea el siguiente:

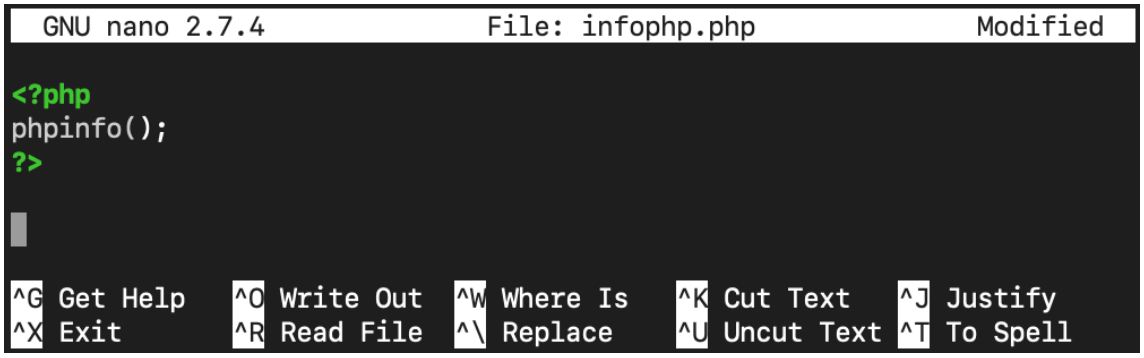
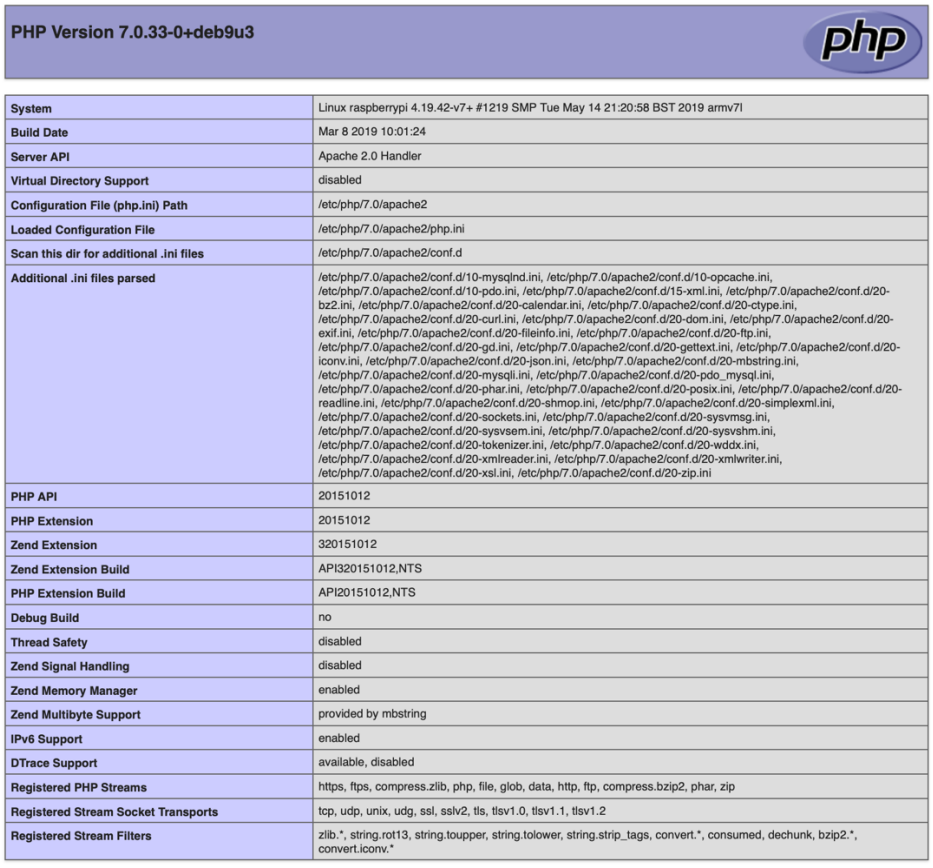


Fig. 17 Captura de pantalla del archivo "infophp.php"

A continuación, en la barra del navegador se escribirá la dirección IP seguida de /infophp.php de tal forma que se podrá ver información sobre la versión de PHP instalada en la *Raspberry*.



PHP Version 7.0.33-0+deb9u3

System	Linux raspberrypi 4.19.42-v7+ #1219 SMP Tue May 14 21:20:58 BST 2019 armv7l
Build Date	Mar 8 2019 10:01:24
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php/7.0/apache2
Loaded Configuration File	/etc/php/7.0/apache2/php.ini
Scan this dir for additional .ini files	/etc/php/7.0/apache2/conf.d
Additional .ini files parsed	/etc/php/7.0/apache2/conf.d/10-mysqlnd.ini, /etc/php/7.0/apache2/conf.d/10-opcache.ini, /etc/php/7.0/apache2/conf.d/10-pdo.ini, /etc/php/7.0/apache2/conf.d/15-xml.ini, /etc/php/7.0/apache2/conf.d/20-bz2.ini, /etc/php/7.0/apache2/conf.d/20-calendar.ini, /etc/php/7.0/apache2/conf.d/20-ctype.ini, /etc/php/7.0/apache2/conf.d/20-curl.ini, /etc/php/7.0/apache2/conf.d/20-dom.ini, /etc/php/7.0/apache2/conf.d/20-exif.ini, /etc/php/7.0/apache2/conf.d/20-fileinfo.ini, /etc/php/7.0/apache2/conf.d/20-ftp.ini, /etc/php/7.0/apache2/conf.d/20-gd.ini, /etc/php/7.0/apache2/conf.d/20-gettext.ini, /etc/php/7.0/apache2/conf.d/20-iconv.ini, /etc/php/7.0/apache2/conf.d/20-json.ini, /etc/php/7.0/apache2/conf.d/20-mbstring.ini, /etc/php/7.0/apache2/conf.d/20-mysqli.ini, /etc/php/7.0/apache2/conf.d/20-pdo_mysql.ini, /etc/php/7.0/apache2/conf.d/20-phar.ini, /etc/php/7.0/apache2/conf.d/20-posix.ini, /etc/php/7.0/apache2/conf.d/20-readline.ini, /etc/php/7.0/apache2/conf.d/20-shmop.ini, /etc/php/7.0/apache2/conf.d/20-simplexml.ini, /etc/php/7.0/apache2/conf.d/20-sockets.ini, /etc/php/7.0/apache2/conf.d/20-sysvmsg.ini, /etc/php/7.0/apache2/conf.d/20-sysvsem.ini, /etc/php/7.0/apache2/conf.d/20-sysvshm.ini, /etc/php/7.0/apache2/conf.d/20-tokenizer.ini, /etc/php/7.0/apache2/conf.d/20-wddx.ini, /etc/php/7.0/apache2/conf.d/20-xmlreader.ini, /etc/php/7.0/apache2/conf.d/20-xmlwriter.ini, /etc/php/7.0/apache2/conf.d/20-xsl.ini, /etc/php/7.0/apache2/conf.d/20-zip.ini
PHP API	20151012
PHP Extension	20151012
Zend Extension	320151012
Zend Extension Build	API320151012,NTS
PHP Extension Build	API20151012,NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	disabled
Zend Memory Manager	enabled
Zend Multibyte Support	provided by mbstring
IPv6 Support	enabled
DTrace Support	available, disabled
Registered PHP Streams	https, ftps, compress.zlib, php, file, glob, data, http, ftp, compress.bzip2, phar, zip
Registered Stream Socket Transports	tcp, udp, unix, udg, ssl, sslv2, tls, tlsv1.0, tlsv1.1, tlsv1.2
Registered Stream Filters	zlib.*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, dechunk, bzip2.*, convert.iconv.*

Fig. 18 Captura de pantalla sobre la información de la versión de PHP instalada correctamente

En cambio, para comprobar la correcta instalación de Apache, bastará con buscar en el navegador la dirección IP de nuestro servidor seguido de "/index.html".

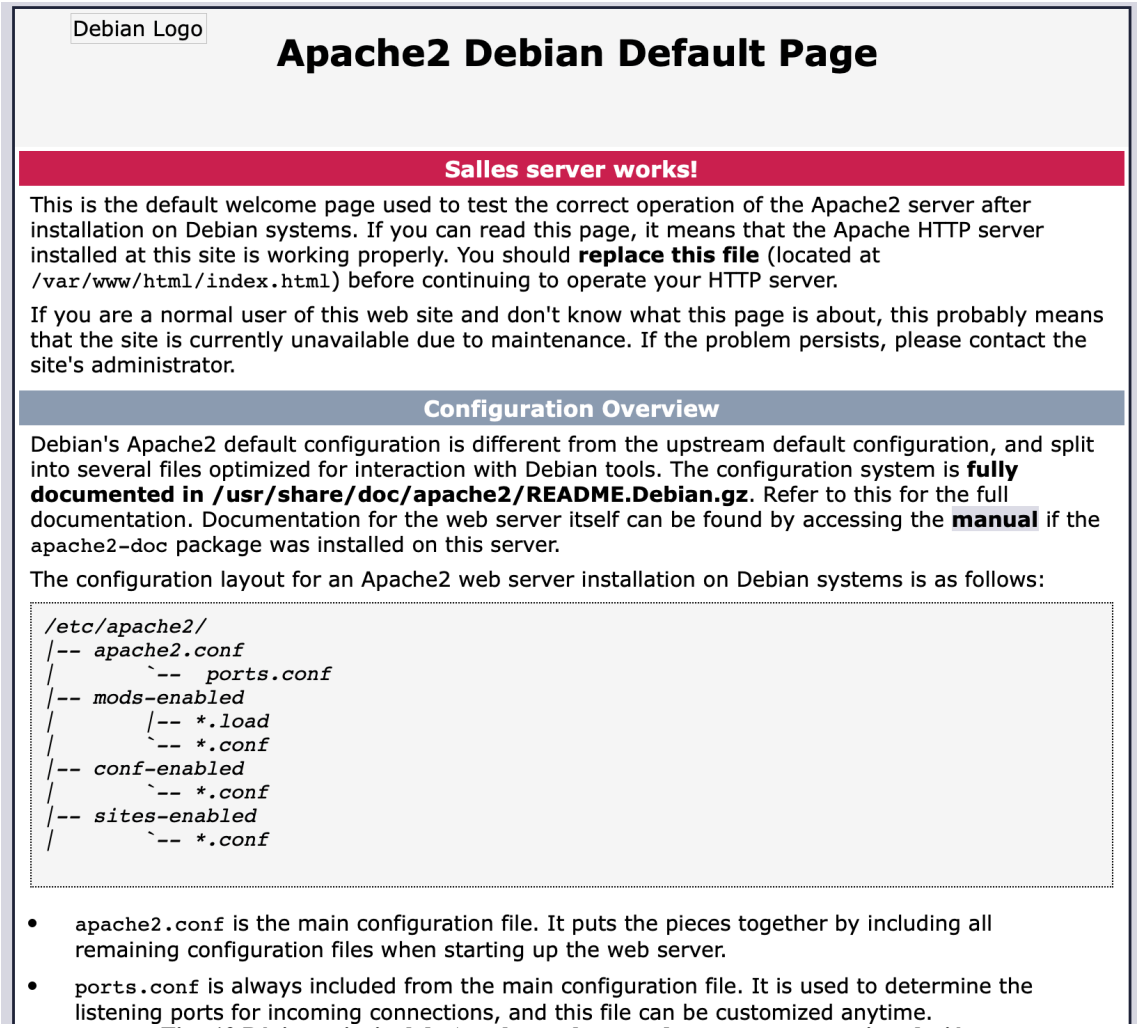


Fig. 19 Página principal de Apache en el navegador tras su correcta instalación

A continuación, se debe instalar un sistema de gestión de base de datos. Como se ha comentado anteriormente y con el fin de aprovechar todo el potencial que ofrece PHP, se utilizará MySQL. Para su instalación, se debe ejecutar el comando "sudo apt-get install mysql-server". Durante su instalación se pedirá la contraseña del usuario *root* que, si todavía no ha sido cambiada, seguirá siendo la contraseña por defecto "raspberry".

Ahora sí que es posible comenzar a crear la base de datos mediante línea de comandos, pero con el fin de facilitar la gestión de la base de datos se instalará phpmyadmin mediante el comando "sudo apt-get install phpmyadmin". Phpmyadmin es una herramienta que ayuda a crear las bases de datos mediante una página web con una interfaz gráfica que sustituye a las líneas de comandos con las que se trabaja. Durante la instalación de esta herramienta se solicita tanto el tipo de servidor que se ha instalado como la contraseña escogida a la hora de instalar MySQL. Además, escogeremos un nombre de usuario y contraseña para poder acceder más tarde a la gestión de la

base de datos. Para confirmar que la instalación se ha completado correctamente, bastará con escribir en el navegador web la dirección IP fija que hemos asignado al servidor seguida de "/phpmyadmin".

Fig. 20 Página de inicio para el servicio phpmyadmin

Por último, será necesario poder acceder al servidor desde cualquier parte, ya sea desde dentro de nuestra red como desde fuera. Por eso, es necesario configurar los puertos del *router* que proporciona el acceso a internet y asignar un nombre de dominio virtual a la dirección IP estática de la red donde estará alojado el servidor.

En cuanto a los puertos, se necesita acceder a la configuración del *router*. Por defecto, se escribirá en el navegador de un ordenador conectado a la misma red del servidor la dirección http://192.168.1.1 y se accederá a la configuración de este mediante la contraseña proporcionada por el administrador de la red. Una vez dentro de la configuración, se habilitarán los puertos 80 y 22 para los protocolos UDP y TCP.

Existen múltiples páginas que ofrecen servicios donde obtener nombres de dominios virtuales y asignarlos a direcciones IP, un ejemplo es www.no-ip.com. Este servicio es escogido debido a que cuenta con la posibilidad de utilizar varios subdominios y dar de alta varios nombres de forma totalmente gratuita. Para el uso de este servicio solo será necesario crear una cuenta en la página

web y asignar a nuestro nombre de dominio la dirección IP pública a la que esta conectado el *router*.

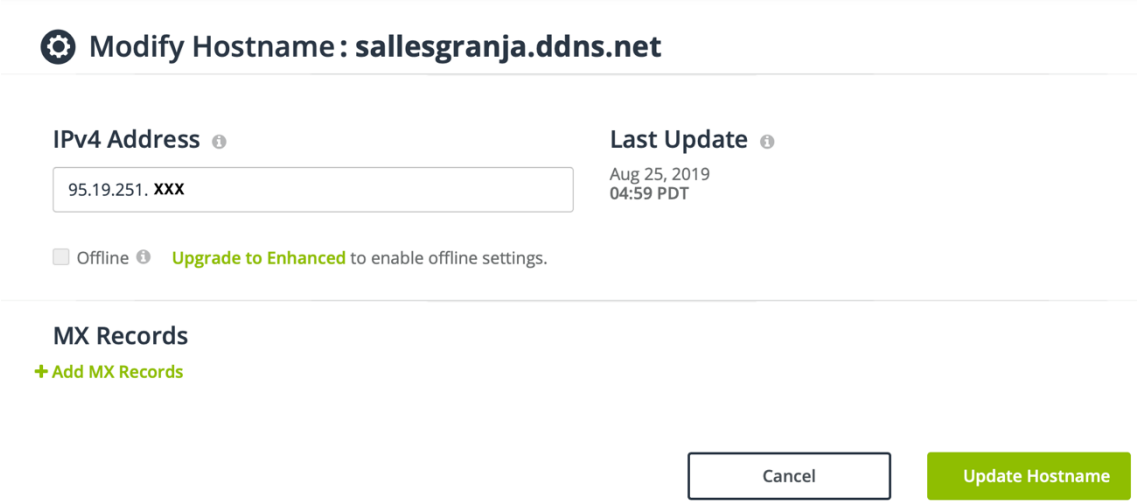


Fig. 21 Captura de pantalla del servicio no-ip

Se debe tener en cuenta que la dirección IP pública del *módem* podría variar, por lo que se debe configurar la *Raspberry* para que automáticamente detecte la dirección IP pública del *módem* y sea capaz de actualizarla en el servicio de no-ip en caso de que se haya modificado. Para ello será necesario crear una carpeta donde se descargará, se descomprimirá y se instalará el paquete de no-ip a través de los siguientes comandos:

1. mkdir no-ip
2. cd no-ip
3. wget http://www.no-ip.com/client/linux/noip-duc-linux.tar.gz
4. tar -zxvf noip-duc-linux.tar.gz
5. cd noip-2.1.9-1/
6. make
7. sudo make install

Durante la instalación, se solicita el nombre de usuario y la contraseña del servicio de no-ip para comprobar el nombre de dominio que se ha dado de alta. En cuanto a la puesta en marcha del servicio no-ip en la *Raspberry* bastaría con iniciarlo mediante el comando "sudo usr/local/bin/noip2" pero, para proteger el servicio en caso de caída del servidor y evitar tener que arrancarlo a mano cada vez que este se reinicie, se creará un archivo encargado de arrancarlo automáticamente cada vez que la *Raspberry* se encienda. Para crear este fichero, se utilizará el comando "sudo nano /etc/init.d/noip2" con el que se abrirá un editor de texto en el cual se pegará la siguiente línea de código:

Fig. 22 Captura de pantalla del fichero noip2

Por último, hay que dotar de los permisos necesarios y añadir a la cola de arranque el archivo creado mediante los comandos “sudo chmod +x /etc/init.d/noip2” y “sudo update-rc.d noip2 defaults”.

5.5 Diseño de la base de datos

En la base de datos será necesario guardar tanto los datos de los usuarios y las diferentes granjas como los valores de las mediciones de cada uno de los sensores y los datos técnicos de cada granja con los que se llevará un control de la cantidad de conejos que se crían en cada banda.

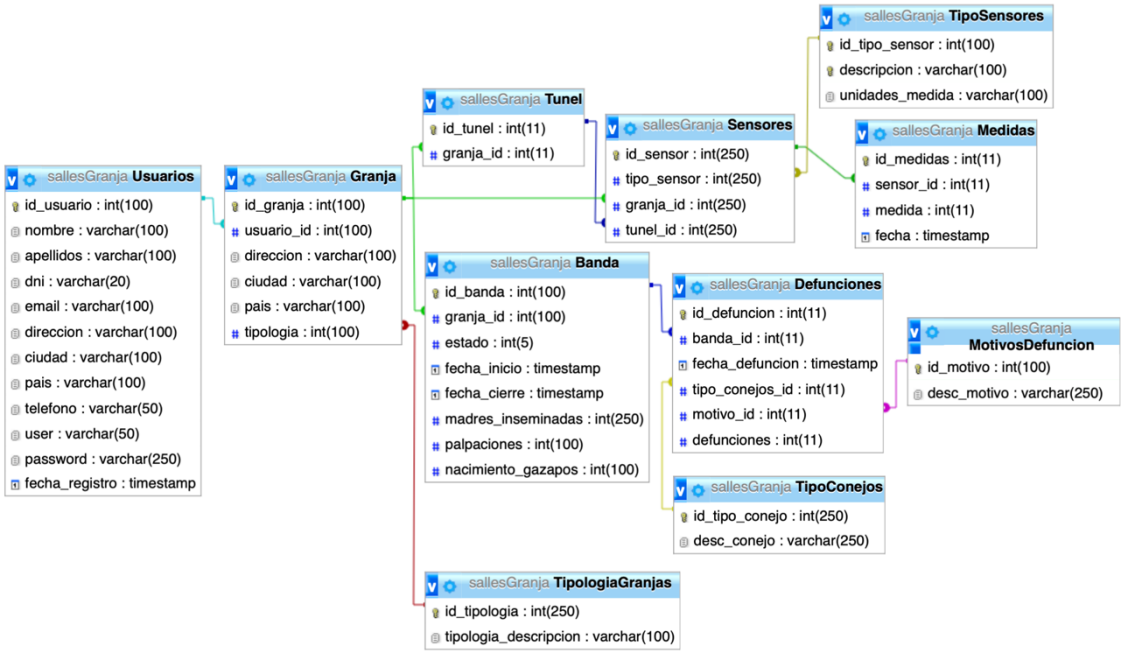


Fig. 23 Esquema relacional de las tablas de la base de datos

En cuanto a cada una de las granjas, su información general como dirección, ciudad, país e incluso de qué tipo de animales es la granja, se guardará en la tabla "Granja". Esta tabla estará relacionada con la tabla "Usuarios" a través del id. de cada uno de los usuarios y donde se guardará la siguiente información: nombre, apellidos, DNI, email, dirección, ciudad, país, teléfono, nombre de usuario para acceder a la app, contraseña y la fecha de cuando se registró. Este diseño permite que varías granjas puedan estar relacionadas con un mismo usuario.

En la tabla "TipologiaGranjas" se asigna un id. a cada uno de los tipos de granjas según los animales y se relaciona con la tabla "Granjas" para poder distinguirlas, aunque este proyecto está centrado únicamente en las granjas cunícolas.

Las granjas de conejos están divididas en túneles donde se separan los diferentes tipos de conejos según su edad. Estos túneles tendrán asignado un id. representativo en la tabla "Tunel" para poder diferenciarlos y estarán relacionados con el id. de la granja a la que pertenecen de tal forma que varios túneles pueden estar relacionados con una misma granja.

En cuanto a los datos que se van a representar en la aplicación, se puede distinguir dos tipos: datos de sensorización y datos técnicos de banda.

Los datos de sensorización son aquellos que se obtienen mediante la red de sensorizado instalada previamente en la granja. Esta red estará compuesta por diferentes tipos de sensores entre los que se pueden encontrar sensores de temperatura, humedad, oxígeno, CO2, sensores ultrasonidos para medir la cantidad de agua que queda disponible o la cantidad de los diferentes piensos que hay en la granja, como el pienso de engorde, maternal o pienso de retirada y, por último, sensores para

conocer si la calefacción está, o no, encendida. Todos estos diferentes tipos se pueden encontrar en la tabla "TipoSensores" donde se les asigna un id., una descripción para diferenciar cada uno de los tipos nombrados y las unidades de medida. Cada una de las granjas puede contar con todos o solo alguno de ellos, por eso, en la tabla "Sensores" se le asigna un id. a cada uno de los dispositivos físicos instalados y se relaciona con el tipo de sensor, granja y túnel al que corresponde. Todas las medidas de la red de sensores se guardarán en la tabla "Medidas" diferenciándolas entre si con un id. por cada lectura del sensor y relacionándola con el dispositivo físico que ha tomado el valor. Además, se guarda también la fecha y hora a la que la misma ha sido añadida a la base de datos.

Como se ha explicado anteriormente, las granjas cunícolas se organizan en bandas. Para poder llevar un control del número de crías que nacen, así como el total de conejos que se llevan al matadero, se ha creado la tabla "Banda" donde se guardarán cada una de ellas y se diferenciarán mediante un id. Cada una de estas bandas estará relacionada con una granja a la que pertenecen y donde se están criando los conejos. También se guardarán datos como la fecha de inicio de cada banda, la fecha de cierre en la que los conejos son llevados al matadero, el total de madres inseminadas, el número de palpaciones que indica cuántas madres se encuentran en periodo de gestación, el número de crías que finalmente nacen y el estado en el que se encuentra la banda.

Además, se llevará un control de la cantidad de defunciones que se puedan producir mediante el transcurso de una banda gracias a la tabla "Defunciones" donde se indicará la banda a la que pertenecen dichos datos, la fecha de cuando se producen las defunciones, el tipo de conejo que fallece para poder diferenciar entre madre y cría y la causa de la baja. Estos motivos estarán descritos en la tabla "MotivosDefuncion" y se relacionarán con la tabla "Defunciones" mediante un identificativo para cada una de las descripciones.

5.6 API REST

En primer lugar, se debe definir lo que es una API. API son las siglas de interfaz de programación de aplicaciones y se define como el conjunto de funciones, métodos y protocolos mediante los cuales se comunican varias aplicaciones entre si. Funciona como si de una librería se tratase a la cual el usuario puede realizar varias peticiones y la API le contesta con la respuesta y un código numérico que indica si todo ha ido correctamente o se ha producido un error y de que tipo de error se trata.

Las siglas REST provienen de 'transferencia de estado representacional' y definen a las interfaces que hacen uso del protocolo HTTP para obtener o manejar los datos en formatos como XML o JSON. Algunas de las características que definen este tipo de servicio son:

- **Protocolo cliente/servidor sin estado.** Esto significa que ni cliente ni servidor necesitan conocer como esta desarrollada la parte contraria posibilitando así que estos dos se puedan desarrollar de forma paralela. Además, no será necesario recordar el estado de la sesión ya que cada una de las peticiones es independiente y contiene toda la información necesaria.
- **Uso de las operaciones básicas de HTTP.** Las cuatro operaciones más importantes del protocolo HTTP son: GET (consultar), POST (crear), PUT (editar), DELETE (eliminar).
- **Objetos manipulados mediante URI.** Las direcciones URI son los elementos mediante los que accederemos a los recursos de nuestra API REST.

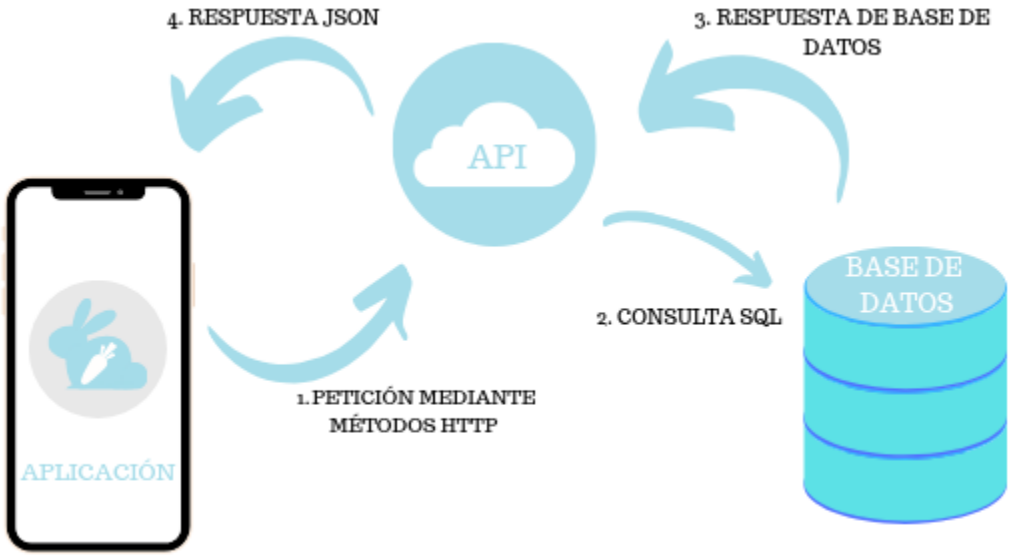


Fig. 24 Estructura API REST

5.6.1 Desarrollo de la API REST para este proyecto

Para el desarrollo de la API REST de este proyecto será necesario el uso del lenguaje PHP con el que se desarrollarán, mediante cualquier editor de texto, como por ejemplo *Visual Studio Code*, todos los ficheros necesarios. Si la *Raspberry* se ha configurado en modo Headless, será necesario también el uso de *FileZilla* para transferir los archivos al servidor de una manera cómoda y sencilla.

FileZilla es el programa libre de código abierto utilizado para acceder de forma remota al servidor a través de los protocolos *FTP* y *SFTP*. Mediante dicho software podemos navegar por los diferentes directorios de una forma muy visual y sencilla. Además, podremos mover todos los archivos desarrollados para la API desde nuestro propio ordenador al servidor con el simple hecho de copiar y pegar los ficheros.

El servicio de la API REST se desarrollará en el directorio /var/www/html/api y, con el fin de organizarlo de la manera más simplificada posible, se crearán 4 directorios dentro de este.

El primero de ellos será el directorio "config" donde se guardarán dos ficheros. En el archivo database.php se guardará toda la configuración básica y necesaria para la conexión con la base de datos y, en el fichero core.php, se guardarán las variables necesarias para crear y decodificar los *tokens*.

El siguiente directorio será el directorio "libs" donde se guardarán todas las librerías descargadas para la generación y decodificación de los *tokens*.

En la carpeta "objects" encontramos varios archivos donde se encuentran las funciones encargadas de realizar las peticiones necesarias a la base de datos.

Uno de estos ficheros es "usuarios.php" donde encontraremos funciones para crear nuevos usuarios, así como obtener los datos de un usuario dado el nombre de dicho usuario.

Con "granjas.php" y su función "getGranjas()" se obtiene tanto el identificativo de las granjas que pertenecen a un usuario dado como las direcciones y las ciudades donde se encuentran dichas granjas.

Otro de los archivos es "tuneles.php" donde encontramos la función "getTuneles()" mediante la cual obtenemos el id de los túneles que hay en una granja dado el número identificativo de dicha granja.

En "medidas.php" se han guardado todas las funciones necesarias para obtener la última medida guardada por cada uno de los sensores, así como una función mediante la cual se obtendrán los datos de los *datasets* necesarios para crear las gráficas en la aplicación.

```
function getTemperatura(){
    $query = "SELECT Tunel.id_tunel, TipoSensores.descripcion, Medidas.medida,
        TipoSensores.unidades_medida
                FROM " . $this->table_name . "
                INNER JOIN Sensores ON Sensores.id_sensor = Medidas.sensor_id
                INNER JOIN TipoSensores ON TipoSensores.id_tipo_sensor = Sensores.tipo_sensor
                INNER JOIN Tunel ON Tunel.id_tunel = Sensores.tunel_id
                WHERE Tunel.id_tunel = ?
                AND TipoSensores.descripcion = 'Temperatura'
                ORDER BY Medidas.fecha DESC LIMIT 1";

    $stmt = $this->conn->prepare($query);
    $stmt->bindParam(1, $this->id_tunel);
    $stmt->execute();
    $row = $stmt->fetch(PDO::FETCH_ASSOC);

    $this->descripcion = $row['descripcion'];
    $this->medida = $row['medida'];
    $this->unidades_medida = $row['unidades_medida'];
}
```

Fig. 25 Función para obtener la última medida del sensor de temperatura

El fichero “bandas.php” tendrá las funciones necesarias para crear bandas, obtener las bandas que pertenecen a una granja, obtener todos los datos relacionados con una banda dado su número identificativo y una función mediante la cual se podrá actualizar bandas.

Por último, el archivo “defunciones.php” contendrá funciones para guardar las muertes de los conejos, para obtener el número total de muertes en una banda según el tipo de conejo y el motivo de la defunción y para obtener los *datasets* necesarios para poder crear las gráficas de defunciones en la app.

El último directorio será la carpeta “usuario” donde se guardarán una serie de ficheros a los que la aplicación se conectará dependiendo de la página o la acción que haya querido realizar el usuario. A su vez, estos archivos importarán algunos de los ficheros del directorio “objects” para poder obtener la información necesaria de la base de datos y una vez obtenida devolverá una respuesta ordenada o un mensaje de error en caso de que este se produzca. Para poder conectarse a estos ficheros, la app tendrá que añadir a la dirección raíz (sallesgranja.ddns.net) seguida de una barra y el nombre del archivo como por ejemplo “sallesgranja.ddns.net/login.php”.

En la primera pantalla de la aplicación se solicita las credenciales al usuario para poder iniciar sesión. Estas credenciales han de ser verificadas para permitirlo o denegarlo. Para ello, cuando el usuario pulsa el botón, la app manda las credenciales a la dirección “/login.php” la cual comprueba en primer lugar que exista algún usuario con el mismo *user*. Si este existe, codifica la contraseña recibida y comprueba que sea la misma que la que está guardada en base de datos. De ser todo esto cierto, se genera y codifica un *token* para devolver al usuario y cuya función será la de comprobar que dicho usuario tiene permiso para mantener la conexión con el servidor ya que ha iniciado sesión anteriormente. Se ha diseñado dicho *token* de tal forma que en principio no tenga fecha de caducidad con el fin de facilitar el uso de la aplicación a los ganaderos.

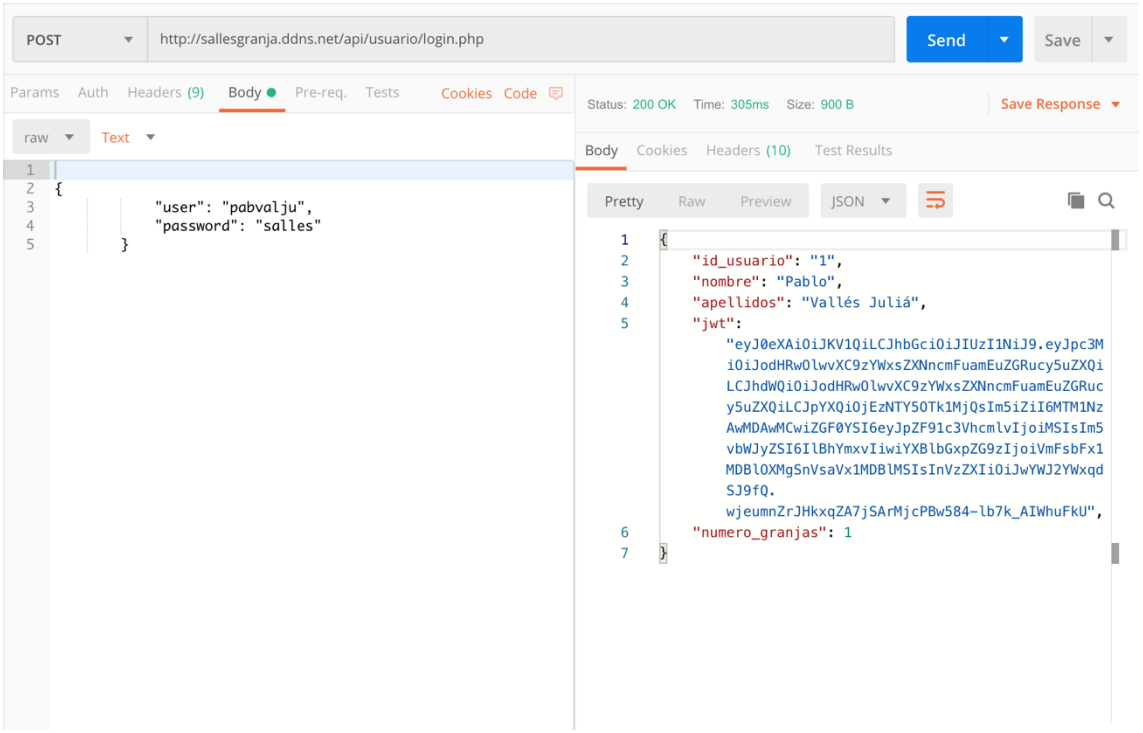


Fig. 26 Comprobación del funcionamiento de "/login.php"

En la siguiente página, donde se selecciona la granja de la cual se quiere obtener la información, la app enviará a "/selectGranjas.php" el id del usuario junto al *token* que previamente ha recibido al iniciar sesión. Este fichero, al igual que el resto de los ficheros, comprobarán en primer lugar que el *token* recibido sea el correcto para ese usuario y esta base de datos. En segundo lugar y con el número identificativo del usuario recibido, se obtienen los IDs de las granjas, las direcciones y las ciudades donde se encuentran todas las granjas relacionadas con dicho usuario para devolverlas de forma ordenada y, más adelante, mostrarlas mediante una lista en la página de la aplicación.

La dirección "/estGen.php" servirá para obtener la información general de cada uno de los túneles de la granja. A este archivo le llegará por parte de la aplicación tanto el *token* como el ID de la granja que ha seleccionado previamente el usuario. Con estos datos, la API obtendrá el número identificativo de cada uno de los túneles relacionados con dicha granja y, una vez obtenidos todos los IDs, se creará y se devolverá como respuesta un *array* compuesto de varios vectores en los cuales se podrá diferenciar el ID del túnel y el último valor guardado de los sensores de temperatura, humedad, oxígeno y CO2 de dicho túnel. En caso de que cualquier túnel tenga instalados varios sensores, por ejemplo, de temperatura, el valor guardado en base de datos sería la media del valor medido por cada uno de ellos y, por lo tanto, el valor devuelto.

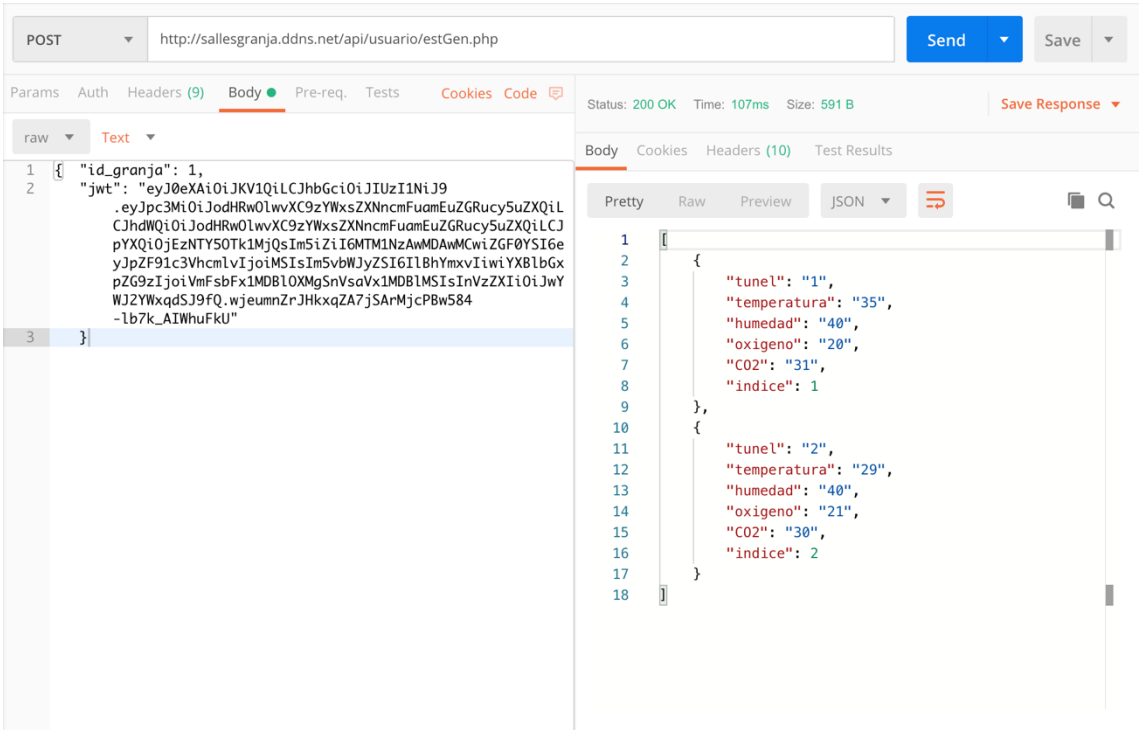


Fig. 27 Captura de pantalla comprobando el funcionamiento de "/estGen.php"

Para observar las medidas de pienso, agua o el estado de la calefacción y el cooling, la aplicación lanzará una petición a "/estPiensoYAgua.php" a la cual, además de enviar el *token* y el ID de la granja, mandará también guardado en una variable llamada "descripcion" la palabra "Agua", "Pienso", "Calefaccion" o "Cooling" según desde que página de la aplicación haya sido lanzada la petición. Si la petición es lanzada desde la página de estado del agua, la API obtendrá el último valor guardado de la cantidad de litros de agua disponibles en el tanque y lo devolverá junto a otra variable llamada sensor en la que se incluirá la descripción del tipo de sensor. Si la API recibe la palabra "Pienso", se obtendrá de la base datos la última medida de cada uno de los tipos de pienso: pienso de engorde, pienso maternal y pienso de retirada. El valor de la medida obtenida por cada uno de los tipos de pienso se guardará en un vector en el que se indicará también de que tipo de pienso se trata. Antes de devolver estos valores, cada uno de los diferentes vectores se juntarán en un único *array* que será devuelto como la respuesta por parte del servidor. Si el usuario está tratando de consultar si la calefacción o el *cooling* están encendidos, la aplicación realizará dos peticiones al servidor. En la primera se mandará la palabra "Calefaccion" y se obtendrá de la base de datos un 1 o un 0 en caso de que esté encendida o apagada la calefacción. En la segunda, la API recibirá la palabra "Cooling" y buscará las últimas medidas de los 4 sensores de *cooling* instalados en la granja para devolver la descripción de cada uno de los sensores con la medida que ha obtenido de la base de datos siendo, un 1 si está encendido y un 0 si está apagado.

Para crear los gráficos, será necesario mandar a "/getDatasets.php" el *token*, el id de la granja, un vector con los sensores y otro con las fechas de las que se quiere obtener las medidas. El servidor

irá recorriendo en primer lugar el vector de los sensores y, por cada uno de los sensores que contenga dicho vector, obtendrá el valor medio de las medidas tomadas en cada una de las fechas recibidas en el otro vector. Todas las medidas de cada uno de los sensores se guardarán en un vector junto con otra variable que indicará a que sensor corresponden las medidas. Y finalmente, toda la información de cada uno de los sensores se reunirá en un único vector que se devolverá a la aplicación como respuesta.

Cada vez que el usuario accede al apartado de datos técnicos de banda, la aplicación solicita al servidor información básica de cada una de las bandas que pertenecen a la granja. Para ello, la app se conecta con "/getBandas.php" que solo necesita del *token* y el ID de la granja para devolver un vector por cada banda relacionada con la granja en el que se encuentran datos como el número identificativo, el estado de la banda, la fecha de inicio y la fecha de cierre de la banda.

Si el usuario selecciona una banda ya existente y esta se encuentra en estado finalizado o, una vez dentro de cualquier banda se desea visualizar el resumen de la banda, la aplicación lanzará una petición a "/resumenBanda.php" indicándole el *token* del usuario y el ID de la banda seleccionada. Con estos dos datos, el servidor obtendrá el estado de la banda, la fecha de inicio y de cierre, el número de madres inseminadas, el número de palpaciones, el total de gazapos nacidos y las defunciones de crías y madres pertenecientes a la banda.

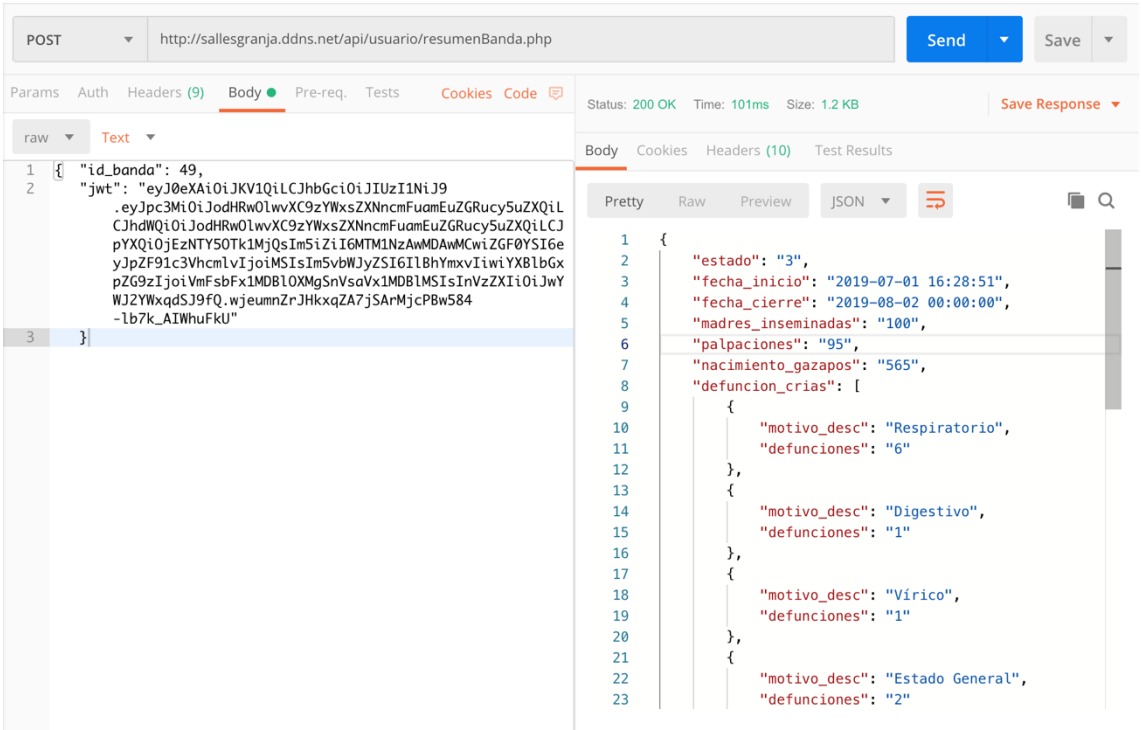


Fig. 28 Captura de pantalla comprobando el funcionamiento de "/resumenBanda.php"

Si en lugar de escoger una banda se desea iniciar una nueva, el usuario deberá indicar la fecha de inicio y el número total de conejas inseminadas. Estos datos junto con el *token* y el ID de la granja se mandará a "/crearBanda.php" para que el servidor se encargue de crear un nuevo registro en la base de datos. Si este registro se ha completado correctamente, la API devolverá un vector con toda la información de la banda donde se incluirá el número identificativo de esta banda que se acaba de crear.

Cuando se inicia una nueva banda, lo primero que se puede guardar es las palpaciones realizadas y el número de conejos que nacen. Para ello, se envía tanto el ID de la banda y el *token* como el número de palpaciones y/o el número de nacimientos a la dirección "/savePalpaciones.php". Este es el encargado de actualizar los datos de la banda de tal forma que, si no se han registrado anteriormente ni palpaciones ni nacimientos, los datos recibidos son guardados directamente. Pero, si esta banda ya contaba con algún valor en estos campos, lo que se hará será sumar el valor recibido con el valor almacenado y actualizar el dato. En caso de que esta acción se complete de forma satisfactoria, el mensaje de respuesta será "Banda actualizada correctamente." mientras que, si ocurre cualquier error, el mensaje será "Error al actualizar banda. Por favor, inténtelo de nuevo más tarde.".

Por otro lado, también se pueden contabilizar el número de defunciones que hay durante el periodo que dura una banda. Para ello se debe rellenar la página de defunciones anotando la fecha en la que se han registrado y, se debe anotar en las diferentes casillas de las diferentes enfermedades el número de muertes que ha habido. Una vez recogidos todos estos datos, la aplicación los ordenará en un vector con diferentes variables de tal forma que el nombre de la variable sea el motivo o la enfermedad y, el valor de esta sea el número de fallecimientos. Estos datos serán enviados a "/saveDefunciones". En ese vector se incluirá el *token*, el ID de la banda, la fecha y el tipo de conejo siendo 1 el número que identifica a los gazapos y 2 a las madres. El servidor recogerá en primer lugar la fecha, el tipo de conejo y la banda e irá recorriendo el vector recibido de tal forma que las variables con nombre de enfermad o motivo de fallecimiento que no estén vacías provocarán que la API cree un nuevo registro de defunciones en la base de datos indicando la fecha, el motivo y el número de fallecimientos.

Una vez finaliza cada una de las etapas de la banda, debemos cerrar la etapa y actualizar el estado de la banda. Para ello, se ha desarrollado "/cerrarEtapa.php" donde se envía el *token*, el ID de la banda, el estado y la fecha de cierre. Cabe decir que la fecha de cierra se enviará siempre a *null* excepto cuando se trate del cierre por completo de la banda en cuyo caso si que se añadirá la fecha del día en que se cierra. En el caso del estado de la banda, se mandará un 1 cuando se crea una banda nueva, un 2 si se pasa del estado inicial o estado de palpaciones al estado actualizar banda y un 3 si se pasa al último de los estados y, por lo tanto, se finaliza con la banda. El servidor, una vez recibe estos datos, comprueba si la fecha esta a *null* y en caso de que este a *null* solo actualiza el dato del estado y, si dispone de una fecha, actualiza también dicho valor.

Capítulo 6. Pliego de condiciones

En primer lugar, y lo más importante para poder llevar a cabo este proyecto es la mano de obra de, como mínimo, un programador con el conocimiento de todos los lenguajes citados a lo largo de este documento. Si se desea reducir el tiempo en el que se desarrolla tanto la aplicación como el servidor con la base de datos, haría falta la ayuda de algunos programadores más.

Este proyecto tampoco tendría sentido sin la previa instalación de sensores en las granjas para lo que se necesitaría una persona encargada de la instalación de estos y otra encargada del desarrollo del código para recoger todos los datos, así como comprobar su correcto funcionamiento. En cuanto a la red de sensores, haría falta un mínimo de 14 sensores (2 de temperatura, 2 de humedad, 2 de oxígeno, 2 de CO2, ultrasonidos para el pienso de engorde, pienso maternal, pienso de retirada y el agua, calefacción y cooling) y un total de 30 aconsejablemente. También serían necesarias varias placas de Arduino por granja.

Además, por cada granja sería necesaria una *Raspberry* o un pequeño servidor encargado de subir todos los datos recogidos por la red de sensores a un servidor general en el que se almacenaría toda la información de cada una de las granjas.

Por lo tanto, también sería necesario un servidor de calidad y con gran almacenamiento donde se pudieran guardar todos los datos de cada una de las granjas y al cual se pudieran conectar numerosos usuarios de forma simultanea sin provocar que este acabar colapsado.

También serían necesarios varios ordenadores en los cuales los programadores pudieran llevar a cabo los desarrollos necesarios para la puesta en marcha de este proyecto.

Por último, se precisaría de las licencias de desarrollo de Android o Apple según la empresa escoja. La licencia de Android cuesta alrededor de 22,55 € y no contiene fecha de caducidad mientras que la de Apple es de aproximadamente 89,28 € y es necesario renovarla cada año.

Capítulo 7. Resultados y conclusiones

Para el desarrollo de este proyecto se ha llevado a cabo un estudio junto a la empresa Hermi, interesada en este trabajo por las innovaciones que esta aplicación aporta a su sector. Esta empresa apuesta por este proyecto puesto que ve una clave de futuro que puede mejorar considerablemente su producción y el funcionamiento de las granjas.

Tras el desarrollo de la aplicación y las previsiones de Hermi, se obtienen las siguientes conclusiones:

La introducción de la revolución 4.0 en el terreno de las granjas provoca una clara mejoría ya que mediante la sensorización de estas se obtiene un mayor cuidado de los animales provocando así un aumento de la producción. Además, se simplifica el control de stock que hay en cada una de las granjas provocando que cada una disponga de los piensos y agua justos y necesarios para poder conservarlos con la máxima calidad.

Otro punto a favor de esta incorporación es la posibilidad de observar mediante gráficos los datos ambientales recogidos durante un intervalo de tiempo, haciendo que se pueda mejorar el clima interno de la granja para un mayor bienestar de los animales. Además, comparar datos ambientales con la cantidad de defunciones también mediante gráficas, hace mucho más sencilla la opción de aumentar la productividad.

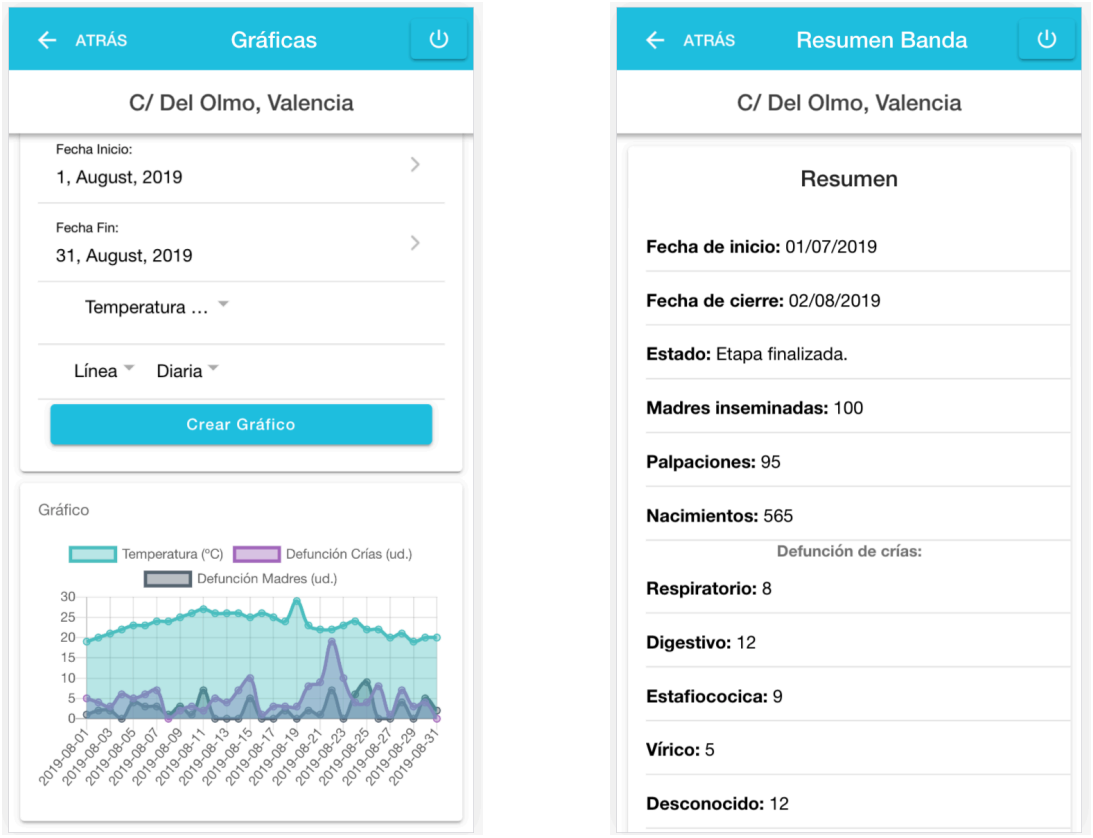


Fig. 29 Capturas de pantalla de comparación entre temperatura y defunciones (izq.) y resumen de banda (drcha.)

Capítulo 8. Propuestas de mejora

Dado el interés mostrado por la empresa Hermi, se pretende seguir evolucionando la aplicación de tal forma que esta pueda cumplir las peticiones de dicha integradora. Una de las propuestas para mejorar la funcionalidad sería la introducción de notificaciones para el usuario en caso de anomalías en las granjas como, por ejemplo, una temperatura excesiva. Estas alertas no han podido ser añadidas en el desarrollo de este proyecto debido a que para poder hacer uso de estas notificaciones en el desarrollo de aplicaciones para el sistema operativo iOS es necesario poseer una cuenta de desarrollador Apple cuyo coste es de 99 libras al año. Además, de la mano de este problema, surgió la idea de controlar con el dispositivo móvil la calefacción para encender o apagar, e incluso escoger la temperatura deseada.

Otra de las mejoras que se propone para un futuro es la automatización de las medidas del pienso de tal forma que cuando se estén agotando las reservas, se rellene un albarán de forma automática y se mande al veterinario para que este confirme el pedido y así mejorar y simplificar los pedidos de pienso y materiales.

Para mejorar la velocidad y la calidad de conexión a la hora de obtener información de la base de datos, se debería sustituir la Raspberry por un servidor con las características mínimas para que se acople a las necesidades según los usuarios que se desee albergar. En este caso no se ha hecho esta sustitución puesto que se trata de un primer prototipo.

También sería necesario añadir una página en la aplicación y un apartado en la base de datos donde se recojan los pesos de los conejos durante diferentes etapas para poder mejorar la producción de kilos de carne que produce cada granja y así poder contar de una forma mucho más rápida y sencilla la cantidad de kilos de carne en vivo que finalmente son llevados al matadero. Este punto no ha sido añadido a este proyecto ya que actualmente estas mediciones son hechas a mano, pero se está tratando de diseñar un dispositivo con el fin de introducirlo en las jaulas de tal forma que no sea necesario medir a mano y se obtengan datos mucho más precisos.

Por último, se podría añadir una página de recuperación de contraseña. A esta página se accedería mediante un hipervínculo en la página de *login* y una vez allí ingresaríamos el correo del usuario y solicitaríamos la recuperación de su clave. Tras este paso, el usuario recibiría en su correo o bien una nueva contraseña o bien un enlace en el que poder escribir la clave que se desee. Esta segunda opción conllevaría tener que diseñar una página web para que el usuario ingrese la contraseña deseada y poder guardarla en base de datos.

Capítulo 9. Bibliografía

Abrir una consola de comandos remota a Raspberry Pi con SSH. *Luis Llamas* [en línea] 2018. Disponible en: https://www.luisllamas.es/raspberry-pi-ssh/

ALVAREZ, M. 2018. Parámetros en las rutas Angular. *DesarrolloWeb.com* [en línea]. [Consulta: 5 julio 2019]. Disponible en: https://desarrolloweb.com/articulos/parametros-rutas-angular.html.

Angular 7 - User Registration and Login Example & Tutorial | Jason Watmore's Blog. *Jasonwatmore.com* [en línea] 2018. Disponible en: https://jasonwatmore.com/post/2018/10/29/angular-7-user-registration-and-login-example-tutorial#app-routing-ts

Autenticación con JSON Web Tokens - Oscar Blancarte - Software Architecture. *Oscar Blancarte - Software Architecture* [en línea] 2017. Disponible en: https://www.oscarblancarteblog.com/2017/06/08/autenticacion-con-json-web-tokens/

CodePen. *CodePen* [en línea] 2019. Disponible en: https://codepen.io/search/pens?q=loading%252520gradient%252520animation&page=1&order=popularity&depth=everything

Como crear un servidor web con Raspberry casero【Actualizado 2019】. *Pedro Pablo Moral* [en línea] 2019. Disponible en: https://www.pedropablomoral.com/raspberrypi/proyectos/servidor-casero/

Como instalar Phpmyadmin en Raspberry. *Alteageek, tutoriales, raspberry pi y cisco, en español* [en línea] 2016. Disponible en: *https://alteageek.com/2016/12/30/como-instalar-phpmyadmin-en-raspberry/*

Configurar IP estática en Raspberry Pi. *Luis Llamas* [en línea] 2018. Disponible en: https://www.luisllamas.es/raspberry-pi-ip-estatica/

Configura tu Raspberry Pi para acceder remotamente usando No-IP. *Patolin.com* [en línea] 2013. Disponible en: http://patolin.com/2013/02/08/configura-tu-raspberry-pi-para-acceder-remotamente-usando-no-ip/

Cómo configurar Raspberry Pi sin monitor ni teclado. *Luis Llamas* [en línea] 2018. Disponible en: https://www.luisllamas.es/como-configurar-raspberry-pi-sin-monitor-ni-teclado/

GRIMM, S. 2019. Navigating the Change with Ionic 4 and Angular Router. *The Ionic Blog* [en línea]. Disponible en: https://ionicframework.com/blog/navigating-the-change-with-ionic-4-and-angular-router/.

Curso PHP MySql. Encriptando Password. Vídeo 68. *YouTube* [en línea] 2019. Disponible en: https://www.youtube.com/watch?v=UsfOT0esKBk&app=desktop

HERRERA, F., 2019. *Instalaciones necesarias para seguir esta playlust* [en línea]. video. 2019. S.l.: s.n. Disponible en: https://www.youtube.com/watch?v=RWCb_ARrSxM&list=PLCKuOXG0bPi2EGYmUq7eidFV8A95xTjEx&index=1.

How To Create A Simple REST API in PHP - Step By Step Guide!. *CodeOfaNinja* [en línea] 2017. Disponible en: https://www.codeofaninja.com/2017/02/create-simple-rest-api-in-php.html

Hybrid App Developers: Don't Store Your User's Passwords. *Joshmorony.com* [en línea] 2019. Disponible en: https://www.joshmorony.com/hybrid-app-developers-dont-store-your-users-passwords/

Ionic Framework. *Ionic Framework* [en línea] Disponible en: https://ionicframework.com/blog/navigating-the-change-with-ionic-4-and-angular-router/

JAMALUDIN, D. 2017. Creating Beautiful Charts Easily using Ionic 3 and Angular 4. *Djamware.com* [en línea]. Disponible en: https://www.djamware.com/post/598953f880aca768e4d2b12b/creating-beautiful-charts-easily-using-ionic-3-and-angular-4.

MOLINA, N. 2019. Gráficos con Ngx-charts. *ion-book* [en línea]. [Consulta: 12 julio 2019]. Disponible en: https://blog.ng-classroom.com//blog/ionic2/ngrx-charts-bars/.

MORONY, J. 2019. When to Use Providers / Services / Injectables in Ionic. *Joshmorony.com* [en línea]. Disponible en: https://www.joshmorony.com/when-to-use-providersservicesinjectables-in-ionic/.

Navegación en Ionic 4. *Medium* [en línea] 2019. Disponible en: https://medium.com/@josephat94/navegación-en-ionic-4-802d42e2d263

Poner la dirección IP fija en Raspbian versiones recientes. *Raspberry para torpes* [en línea] 2016. Disponible en: https://raspberryparatorpes.net/instalacion/poner-la-direccion-ip-fija-en-raspbian-pixel/

REST API Authentication Example in PHP - JWT Tutorial -. *CodeOfaNinja* [en línea] 2019. Disponible en: https://www.codeofaninja.com/2018/09/rest-api-authentication-example-php-jwt-tutorial.html

REVILLA, E. 2019. Tutorial de Ionic - Peticiones http - API REST - Reviblog. *Reviblog* [en línea]. Disponible en: https://reviblog.net/2017/06/19/tutorial-de-ionic-comunicaciones-http-api-rest/.

SHOKEEN, M. 2017. Comenzando Con Chart.js: Gráficas de Línea y Barra. *Code Envato Tuts+* [en línea]. Disponible en: https://code.tutsplus.com/es/tutorials/getting-started-with-chartjs-line-and-bar-charts--cms-28384.

THEORY, G. 2019. Tutorial Raspberry Pi – 9. Servidor FTP. *Geeky Theory* [en línea]. Disponible en: https://geekytheory.com/tutorial-raspberry-pi-9-servidor-ftp.

Tutorial: Cómo crear una animación de carga de contenido estilo Facebook. *platzi.com* [en línea] 2017. Disponible en: https://platzi.com/blog/tutorial-como-crear-una-animacion-de-carga-de-contenido-tipo-facebook/

Using JSON Web Tokens (JWT) for Custom Authentication in Ionic 2: Part 1. *Joshmorony.com* [en línea] 2018. Disponible en: https://www.joshmorony.com/using-json-web-tokens-jwt-for-custom-authentication-in-ionic-2-part-1/

VAQUERO, E. 2018. Como crear componentes personalizados con Ionic - Reviblog. *Reviblog* [en línea]. Disponible en: https://reviblog.net/2018/02/19/crear-componentes-personalizados-ionic/.

Capítulo 10. Anexo

A continuación, se anexa una pequeña parte del código para el desarrollo del proyecto. Por parte de la aplicación podemos ver lo necesario para el desarrollo de únicamente dos páginas de toda la aplicación: la página de inicio de sesión y la página de estado general. Por la parte del servidor, podemos ver el programa escrito en PHP para confirmar el inicio de sesión y obtener los datos que la aplicación solicita en el momento que el usuario accede al apartado "Estado general". Se han escogido únicamente estas dos páginas porque representan parte de la complejidad del proyecto sin exceder de forma exagerada en el tamaño propuesto para la extensión de la memoria.

10.1 Código de la aplicación

10.1.1 Página de inicio de sesión

10.1.1.1 Inicio.page.html

```
<ion-header>
  <ion-toolbar>
    <ion-title class="ion-text-center" text-capitalize>salles</ion-title>
  </ion-toolbar>
</ion-header>

<ion-content>
  <div class="ion-text-center"
       style="margin-top: 3%; margin-bottom: 2%">
    <img width="150"  src="assets/icon/favicon.png">
  </div>

  <form #formulario="ngForm" (ngSubmit)="onSubmit()">
    <ion-list class="ion-text-center">
      <!--Usuario-->
      <ion-item>
        <ion-icon slot="start" name="person"></ion-icon>
        <ion-label position="floating" text-capitalized>
          Usuario:
        </ion-label>
        <ion-input type="text"
                   name="username"
                   [(ngModel)]="usuario.username"
                   required>
        </ion-input>
      </ion-item>
      <!--Password-->
      <ion-item>
        <ion-icon slot="start" name="lock"></ion-icon>
        <ion-label position="floating" text-capitalized>
          Contraseña:
```

```
        </ion-label>
        <ion-input type="password"
                name="password"
                [(ngModel)]="usuario.password"
                required>
        </ion-input>
      </ion-item>

      <!--Boton Login-->
      <div>
       <ion-button [disabled]="formulario.invalid || loading"
          type="submit">
        Iniciar sesión
       </ion-button>
       <div *ngIf="loading">
        <ion-spinner name="lines"></ion-spinner>
       </div>
      </div>

    </ion-list>
  </form>

</ion-content>
```

10.1.1.2 Inicio.page.ts

```
import { Component, OnInit } from '@angular/core';
import { Router } from '@angular/router';
import { AuthenticationService, ToastService } from '../../_services';
import { User } from 'src/app/_models';

@Component({
  selector: 'app-inicio',
  templateUrl: './inicio.page.html',
  styleUrls: ['./inicio.page.scss'],
})
export class InicioPage implements OnInit {
  loading = false;
  returnUrl: string = '/select-granja';
  usuario = {
    username: '',
    password: ''
  };
  respuesta: User;

  constructor(
    private router: Router,
    private authenticationService: AuthenticationService,
    private toast: ToastService) {
```

```
    // Si el usuario ya tiene la sesión iniciada lo redirigimos a /select-granja
    if (this.authenticationService.currentUserValue) {
      this.router.navigate([this.returnUrl]);
    }
  }

  ngOnInit() { }

  onSubmit() {
    this.loading = true;
    this.authenticationService.login(this.usuario.username, this.usuario.password)
      .subscribe(data => {
        this.respuesta = data;
        if (this.respuesta.jwt != null) {
          this.router.navigate([this.returnUrl]);
        }
        this.loading = false;
      }, error => {
        this.loading = false;
        this.toast.presentToast("Usuario o contraseña incorrectos.");
      });
  }
}
```

10.1.2 Servicio de Autenticación

```
import { Injectable } from '@angular/core';
import { HttpClient, HttpHeaders } from '@angular/common/http';
import { BehaviorSubject, Observable } from 'rxjs';
import { map } from 'rxjs/operators';

import { User } from '../_models';
import { GranjasService } from './granjas.service';
import { ToastService } from './toast.service';

@Injectable({ providedIn: 'root' })
export class AuthenticationService {
    private currentUserSubject: BehaviorSubject<User>;
    public currentUser: Observable<User>;
    apiUrl = 'http://sallesgranja.ddns.net/api/usuario';
    datos: any;
    user: any;

    constructor(
        private http: HttpClient,
        private granjaService: GranjasService,
        private toast: ToastService ) {
        this.currentUserSubject = new
BehaviorSubject<User>(JSON.parse(localStorage.getItem('currentUser')));
        this.currentUser = this.currentUserSubject.asObservable();
    }
```

```
  public get currentUserValue(): User {
    return this.currentUserSubject.value;
  }

  login(username: string, password: string) {
    this.datos = {
      "user": username,
      "password": password

}
    return this.http.post(this.apiUrl + '/login.php', JSON.stringify(this.datos))
      .pipe(map(data => {
        this.user = data;
        if (this.user.jwt != null) {
          localStorage.setItem('currentUser', JSON.stringify(this.user));
          this.currentUserSubject.next(this.user);
        }
        return this.user;
      }, error => {
        this.toast.presentToast("Usuario o contraseña incorrectos.");
      }));
  }

  logout() {
    // Eliminamos los datos del usuario de la base local
    localStorage.removeItem('currentUser');
    this.currentUserSubject.next(null);
    // Eliminamos los datos de la granja de la base local
    this.granjaService.logout();
    this.granjaService.removeBanda();
  }
}
```

10.1.3 Página estado general

10.1.3.1 Estado-general.page.html

```
<app-cabecera titulo="general"></app-cabecera>

<ion-content>
  <div *ngIf="loading" class="ion-text-center" style="margin-top: 3%">
    <app-tunel></app-tunel>
  </div>
  <div *ngIf="!loading" class="ion-text-center">
    <div *ngFor="let tunel of tuneles">
      <app-tunel indice={{tunel.indice}} temperatura={{tunel.temperatura}}
        humedad={{tunel.humedad}} oxigeno={{tunel.oxigeno}} co2={{tunel.CO2}}>
      </app-tunel>
    </div>
```

```
  </div>
</ion-content>
```

10.1.3.2 Estado-general.page.ts

```
import { Component, OnInit } from '@angular/core';
import { User, Granja } from 'src/app/_models';
import { Subscription } from 'rxjs';
import { AuthenticationService, GranjasService, ToastService } from 'src/app/_services';

@Component({
  selector: 'app-estado-general',
  templateUrl: './estado-general.page.html',
  styleUrls: ['./estado-general.page.scss'],
})
export class EstadoGeneralPage implements OnInit {
  loading = false;
  currentUser: User;
  currentUserSubscription: Subscription;
  currentFarm: Granja;
  currentFarmSubscription: Subscription;
  jwt: string;
  tuneles: any;
  constructor(
    private authenticationService: AuthenticationService,
    private granjaService: GranjasService,
    private toast: ToastService) {
    this.currentUserSubscription = this.authenticationService.currentUser.subscribe(user => {
      this.currentUser = user;
    });
    this.currentFarmSubscription = this.granjaService.currentFarm.subscribe(farm => {
      this.currentFarm = farm;
    });
  }
  ngOnInit() {
    this.loading = true;
    this.getTuneles();
  }
  getTuneles() {
    this.jwt = this.currentUser.jwt;
    this.granjaService.getEstadoGeneral(this.currentFarm.id_granja, this.jwt)
      .subscribe(data => {
        this.loading = false;
        this.tuneles = data;
      }, error => {
        this.loading = false;
        this.toast.presentToast(error);
```

```
    });
  }
}
```

10.1.4 Componente túnel del estado general

10.1.4.1 Tunel.component.html

```
<ion-card>
  <ion-card-header>
    <ion-card-title> Tunel {{indice}}</ion-card-title>
  </ion-card-header>
  <!--Lista de carga-->
  <ion-list *ngIf="loading">
    <div class="loading-block"></div>
    <div class="loading-block"></div>
    <div class="loading-block"></div>
  </ion-list>
  <!--Lista de datos-->
  <ion-list *ngIf="!loading">
    <ion-item *ngFor="let s of listaSensores">
      <ion-icon slot="start" [name]="s.icono"></ion-icon>
      <ion-label>
        <b>{{s.sensor}}: </b> {{s.medida}} {{s.unidad}}
      </ion-label>
      <ion-button (click)="showGraphs(s.sensor)"><ion-icon slot="end" name="stats"></ion-
icon> </ion-button>
    </ion-item>
  </ion-list>
</ion-card>
```

10.1.4.2 Tunel.component.scss

```
$grey: #dcdcdc;
$light-grey: #f9f9f9;

.loading-block {
  background: linear-gradient(to right, $grey, $light-grey, $grey);
  background: -webkit-linear-gradient(to right, $grey, $light-grey, $grey);
  background-size: 200%;
  height: 30px;
  width: 93%;//200px;
  margin: 3% 3%;
  animation: loading-gradient 2s linear infinite;
}

@keyframes loading-gradient {
  0% {
```

```
    background-position: 0% 50%;
  }
  50% {
    background-position: -100% 50%;
  }
  100% {
    background-position: -200% 50%;
  }
}
```

10.1.4.3 Tunel.component.ts

```
import { Component, OnInit, Input } from '@angular/core';
import { ToastService } from 'src/app/_services';
import { Router } from '@angular/router';

@Component({
  selector: 'app-tunel',
  templateUrl: './tunel.component.html',
  styleUrls: ['./tunel.component.scss'],
})
export class TunelComponent implements OnInit {
  @Input() indice: number;
  @Input() temperatura: string;
  @Input() humedad: string;
  @Input() oxigeno: string;
  @Input() co2: string;
  loading: boolean;

  listaSensores: Sensores[];

  constructor(
    private toast: ToastService,
    private router: Router) { }

  ngOnInit() {
    this.loading = true;
    if (this.indice != null) {
      this.showList();
    }
  }

  showList() {
    if (this.temperatura == '') {
      this.temperatura = "Error de visualización";
      this.toast.presentToast("Error al cargar el valor de 'Temperatura'");
    }
    if (this.humedad == '') {
      this.humedad = "Error de visualización";
```

```
      this.toast.presentToast("Error al cargar el valor de 'Humedad'");
    }
    if (this.oxigeno == '') {
      this.oxigeno = "Error de visualización";
      this.toast.presentToast("Error al cargar el valor de 'Oxígeno'");
    }
    if (this.co2 == '') {
      this.co2 = "Error de visualización";
      this.toast.presentToast("Error al cargar el valor de 'CO2'");
    }
    this.listaSensores = [
      {
        icono: 'thermometer',
        sensor: 'Temperatura',
        medida: this.temperatura,
        unidad: 'ºC'
      },
      {
        icono: 'color-fill',
        sensor: 'Humedad',
        medida: this.humedad,
        unidad: '%'
      },
      {
        icono: 'nuclear',
        sensor: 'Oxígeno',
        medida: this.oxigeno,
        unidad: 'ppm'
      },
      {
        icono: 'nuclear',
        sensor: 'CO2',
        medida: this.co2,
        unidad: 'ppm'
      }
    ];
    this.loading = false;

  }

  showGraphs(sensor){
    this.router.navigate(['/graficas'], { queryParams: { sensor: sensor } });
  }
}
interface Sensores {
  icono: string;
  sensor: string;
  medida: string;
  unidad: string;
}
```

10.2 Código del servidor

10.2.1 Función para inicio de sesión - login.php

```
<?php
    // Cabeceras
    header("Access-Control-Allow-Origin: *");
    header("Content-Type: application/json; charset=UTF-8");
    header("Access-Control-Allow-Methods: POST");
    header("Access-Control-Max-Age: 3600");
    header("Access-Control-Allow-Headers: Content-Type, Access-Control-Allow-Headers, Authorization, X-Requested-With, Origin, Accept");

    // Importamos archivos de conexión con la base de datos
    include_once '../config/database.php';
    include_once '../objects/usuarios.php';
    include_once '../objects/granjas.php';

    // Importamos librerias para generar el token
    include_once '../config/core.php';
    include_once '../libs/BeforeValidException.php';
    include_once '../libs/ExpiredException.php';
    include_once '../libs/SignatureInvalidException.php';
    include_once '../libs/JWT.php';
    use \Firebase\JWT\JWT;

    // Conectamos con la base de datos
    $database = new Database();
    $db = $database->getConnection();

    // Instanciamos el objeto usuario
    $usuario = new Usuario($db);
    $granjas = new Granja($db);

    // Cogemos los datos del método POST
    $data = json_decode(file_get_contents("php://input"));

    // Guardamos los datos en el objeto usuario
    $usuario->user = $data->user;
```

```
$usuario->userExists();
$granjas->usuario_id = $usuario->id_usuario;

// Si el usuario existe en BDD y la contraseña es la correcta generamos el token
if($usuario->id_usuario!=null && password_verify($data->password, $usuario->password)){
   $token = array(
      "iss" => $iss,
      "aud" => $aud,
      "iat" => $iat,
      "nbf" => $nbf,
      "data" => array(
         "id_usuario" => $usuario->id_usuario,
         "nombre" => $usuario->nombre,
         "apellidos" => $usuario->apellidos,
         "user" => $usuario->user
      )
   );
       // Codificamos el token
   $jwt = JWT::encode($token, $key);

   // Obtenemos el número de granjas de dicho usuario
   $stmt = $granjas->getGranjas();
   $num = $stmt->rowCount();

   // Rellenamos el array con la respuesta
   $user_arr = array(
            "id_usuario" => $usuario->id_usuario,
            "nombre" => $usuario->nombre,
            "apellidos" => $usuario->apellidos,
            "jwt" => $jwt,
            "numero_granjas" => $num,
             );
   echo json_encode($user_arr);
   http_response_code(200);
   // Si el login falla:
} else{
   // Repondemos mensaje de error y lo mostramos
   http_response_code(401);
```

```
        echo json_encode(array("message" => "Login failed."));
    }
?>
```

10.2.2 Objeto usuario – usuarios.php

```
<?php
    class Usuario{
        //Conexion y nombre de la tabla
        private $conn;
        private $table_name = "Usuarios";

        //Propiedades
        public $id_usuario;
        public $nombre;
        public $apellidos;
        public $dni;
        public $email;
        public $direccion;
        public $ciudad;
        public $pais;
        public $telefono;
        public $user;
        public $password;
        public $token;
        public $fecha_registro;

        //Conexion con la base de datos
        public function __construct($db){
            $this->conn = $db;
        }

        function userExists(){
            // Query para comprobar si existe el usuario
            $query = "SELECT id_usuario, nombre, apellidos, password
                FROM " . $this->table_name . "
                WHERE user = ?
```

```
                LIMIT 0,1";
            // Preparamos y ejecutamos la query
            $stmt = $this->conn->prepare($query);
            $this->user=htmlspecialchars(strip_tags($this->user));
            $stmt->bindParam(1, $this->user);
            $stmt->execute();
            $num = $stmt->rowCount();

            // Si el usuario existe recogemos los datos de la query
            if($num>0){
               $row = $stmt->fetch(PDO::FETCH_ASSOC);

               $this->id_usuario = $row['id_usuario'];
               $this->nombre = $row['nombre'];
               $this->apellidos = $row['apellidos'];
               $this->password = $row['password'];
            }
        }
    }
?>
```

10.2.3 Objeto granjas – granjas.php

```
<?php
    class Granja{
        //Conexion y nombre de la tabla
        private $conn;
        private $table_name = "Granja";

        //Propiedades de la tabla
        public $id_granja;
        public $usuario_id;
        public $direccion;
        public $ciudad;
        public $pais;
        public $tipologia;
```

```
        //Conexion con la base de datos
        public function __construct($db){
            $this->conn = $db;
        }

            function getGranjas(){
            $query = "SELECT Granja.id_granja, Granja.direccion, Granja.ciudad
                    FROM " . $this->table_name . "
                    INNER JOIN Usuarios ON Usuarios.id_usuario = Granja.usuario_id
                    WHERE Usuarios.id_usuario = ?";

            $stmt = $this->conn->prepare($query);
            $this->usuario_id  = htmlspecialchars(strip_tags($this->usuario_id));
            $stmt->bindParam(1, $this->usuario_id);
            $stmt->execute();

                        return $stmt;
        }
    }
?>
```

10.2.4 Código para obtener los datos en la pantalla estado general – estGen.php

```
<?php
    header("Access-Control-Allow-Origin: *");
    header("Content-Type: application/json; charset=UTF-8");
    header("Access-Control-Allow-Methods: POST");
    header("Access-Control-Max-Age: 3600");
    header("Access-Control-Allow-Headers: Content-Type, Access-Control-Allow-Headers,
Authorization, X-Requested-With");

    include_once '../config/core.php';
    include_once '../libs/BeforeValidException.php';
    include_once '../libs/ExpiredException.php';
    include_once '../libs/SignatureInvalidException.php';
    include_once '../libs/JWT.php';
    use \Firebase\JWT\JWT;
```

```
include_once '../config/database.php';
include_once '../objects/tuneles.php';
include_once '../objects/medidas.php';

// Conexión con base de datos
$database = new Database();
$db = $database->getConnection();

// Instanciamos los objetos
$tuneles = new Tunel($db);
$medidas = new Medida($db);

// Obtenemos el jwt
$data = json_decode(file_get_contents("php://input"));
$jwt=isset($data->jwt) ? $data->jwt : "";

// Si recibimos jwt
if($jwt){
   try {
      $decoded = JWT::decode($jwt, $key, array('HS256'));
      $tuneles->granja_id = $data->id_granja;
            $stmt = $tuneles->getTuneles();
            $num = $stmt->rowCount();
      $i = 1;
      $respuesta_arr = array();

      while($row = $stmt->fetch(PDO::FETCH_ASSOC)){
         extract($row);
         $tuneles->id_tunel = $row['id_tunel'];
         $medidas->id_tunel = $tuneles->id_tunel;

         $medidas->getTemperatura();
         $medidaTemp = $medidas->medida;

         $medidas->getHumedad();
         $medidaHum = $medidas->medida;

         $medidas->getOxigeno();
```

```
            $medidaOxigeno = $medidas->medida;

            $medidas->getCO2();
            $medidaCO2 = $medidas->medida;

            $respuesta = array(
                        "tunel" => $medidas->id_tunel,
                        "temperatura" => $medidaTemp,
                        "humedad" => $medidaHum,
                        "oxigeno" => $medidaOxigeno,
                        "CO2" => $medidaCO2,
                        "indice" => $i

                            );
            array_push($respuesta_arr, $respuesta);
            $i++;
          }
          http_response_code(200);
          echo json_encode($respuesta_arr);

       }catch (Exception $e){
          http_response_code(401);
          echo json_encode(array(
             "message" => "Acceso denegado.",
             "error" => $e->getMessage()
          ));
       }
    }else{
       http_response_code(401);
       echo json_encode(array("message" => "Acceso denegado."));
    }
?>
```

10.2.5 Objeto tuneles – tuneles.php

```
<?php
   class Tunel{
```

```
        //Conexion y nombre de la tabla
        private $conn;
        private $table_name = "Tunel";

        //Propiedades
        public $id_tunel;
        public $granja_id;

        //Conexion con la base de datos
        public function __construct($db){
            $this->conn = $db;
        }

        function getTuneles(){
            $query = "SELECT Tunel.id_tunel FROM " . $this->table_name . "
                    INNER JOIN Granja ON Granja.id_granja = Tunel.granja_id
                    INNER JOIN Usuarios ON Usuarios.id_usuario = Granja.usuario_id
                    WHERE Granja.id_granja = ? ";

            $stmt = $this->conn->prepare($query);
            $stmt->bindParam(1, $this->granja_id);
                $stmt->execute();

            return $stmt;
        }
    }
?>
```

10.2.6 Objeto medidas- medidas.php

```
<?php
    class Medida{
        //Conexion y nombre de la tabla
        private $conn;
        private $table_name = "Medidas";

        //Propiedades
```

```
        public $id_tunel;
    public $id_granja;
    public $medida;
    public $descripcion;
    public $unidades_medida;
    public $sensor;
    public $fecha;

    //Conexion con la base de datos
    public function __construct($db){
      $this->conn = $db;
    }

    function getTemperatura(){
      $query = "SELECT Tunel.id_tunel, TipoSensores.descripcion, Medidas.medida,
TipoSensores.unidades_medida
                                    FROM " . $this->table_name . "
                                    INNER JOIN Sensores ON Sensores.id_sensor
= Medidas.sensor_id
                                    INNER JOIN TipoSensores ON
TipoSensores.id_tipo_sensor = Sensores.tipo_sensor
                                    INNER JOIN Tunel ON Tunel.id_tunel =
Sensores.tunel_id
                                    WHERE Tunel.id_tunel = ?
                                    AND TipoSensores.descripcion = 'Temperatura'
                                    ORDER BY Medidas.fecha DESC LIMIT 1";

      $stmt = $this->conn->prepare($query);
      $stmt->bindParam(1, $this->id_tunel);
        $stmt->execute();
      $row = $stmt->fetch(PDO::FETCH_ASSOC);

      $this->descripcion = $row['descripcion'];
      $this->medida = $row['medida'];
      $this->unidades_medida = $row['unidades_medida'];
    }

        function getHumedad(){
```

```
$query = "SELECT Tunel.id_tunel, TipoSensores.descripcion, Medidas.medida,
TipoSensores.unidades_medida
FROM " . $this->table_name . "
INNER JOIN Sensores ON Sensores.id_sensor = Medidas.sensor_id
INNER JOIN TipoSensores ON TipoSensores.id_tipo_sensor =
Sensores.tipo_sensor
INNER JOIN Tunel ON Tunel.id_tunel = Sensores.tunel_id
        WHERE Tunel.id_tunel = ?
AND TipoSensores.descripcion = 'Humedad'
ORDER BY Medidas.fecha DESC LIMIT 1";

$stmt = $this->conn->prepare($query);
$stmt->bindParam(1, $this->id_tunel);
$stmt->execute();
$row = $stmt->fetch(PDO::FETCH_ASSOC);

        $this->descripcion = $row['descripcion'];
        $this->medida = $row['medida'];
        $this->unidades_medida = $row['unidades_medida'];
}

function getOxigeno(){
$query = "SELECT Tunel.id_tunel, TipoSensores.descripcion, Medidas.medida,
TipoSensores.unidades_medida
FROM " . $this->table_name . "
INNER JOIN Sensores ON Sensores.id_sensor = Medidas.sensor_id
INNER JOIN TipoSensores ON TipoSensores.id_tipo_sensor =
Sensores.tipo_sensor
INNER JOIN Tunel ON Tunel.id_tunel = Sensores.tunel_id
WHERE Tunel.id_tunel = ?
AND TipoSensores.descripcion = 'Oxígeno'
ORDER BY Medidas.fecha DESC LIMIT 1";

$stmt = $this->conn->prepare($query);
$stmt->bindParam(1, $this->id_tunel);
$stmt->execute();
$row = $stmt->fetch(PDO::FETCH_ASSOC);

$this->descripcion = $row['descripcion'];
```

```
            $this->medida = $row['medida'];
            $this->unidades_medida = $row['unidades_medida'];
            }

    function getCO2(){
        $query = "SELECT Tunel.id_tunel, TipoSensores.descripcion, Medidas.medida,
TipoSensores.unidades_medida
        FROM " . $this->table_name . "
        INNER JOIN Sensores ON Sensores.id_sensor = Medidas.sensor_id
        INNER JOIN TipoSensores ON TipoSensores.id_tipo_sensor = Sensores.tipo_sensor
        INNER JOIN Tunel ON Tunel.id_tunel = Sensores.tunel_id
        WHERE Tunel.id_tunel = ?
        AND TipoSensores.descripcion = 'CO2'
        ORDER BY Medidas.fecha DESC LIMIT 1";

        $stmt = $this->conn->prepare($query);
        $stmt->bindParam(1, $this->id_tunel);
        $stmt->execute();
        $row = $stmt->fetch(PDO::FETCH_ASSOC);

        $this->descripcion = $row['descripcion'];
        $this->medida = $row['medida'];
        $this->unidades_medida = $row['unidades_medida'];
    }

    function getEstPienso(){
        $query = "(SELECT TipoSensores.descripcion, Medidas.medida, Medidas.id_medidas
        FROM " . $this->table_name . "
        INNER JOIN Sensores ON Sensores.id_sensor = Medidas.sensor_id
        INNER JOIN TipoSensores ON TipoSensores.id_tipo_sensor = Sensores.tipo_sensor
        INNER JOIN Granja ON Granja.id_granja = Sensores.granja_id
        WHERE Granja.id_granja = ?
        AND TipoSensores.descripcion = 'Pienso engorde'
        ORDER BY Medidas.fecha DESC LIMIT 1)
        UNION (
            SELECT TipoSensores.descripcion, Medidas.medida, Medidas.id_medidas
            FROM " . $this->table_name . "
            INNER JOIN Sensores ON Sensores.id_sensor = Medidas.sensor_id
```

```
            INNER JOIN TipoSensores ON TipoSensores.id_tipo_sensor = Sensores.tipo_sensor
            INNER JOIN Granja ON Granja.id_granja = Sensores.granja_id
            WHERE Granja.id_granja = ?
            AND TipoSensores.descripcion = 'Pienso maternal'
            ORDER BY Medidas.fecha DESC LIMIT 1)
        UNION (
            SELECT TipoSensores.descripcion, Medidas.medida, Medidas.id_medidas
            FROM " . $this->table_name . "
            INNER JOIN Sensores ON Sensores.id_sensor = Medidas.sensor_id
            INNER JOIN TipoSensores ON TipoSensores.id_tipo_sensor = Sensores.tipo_sensor
            INNER JOIN Granja ON Granja.id_granja = Sensores.granja_id
            WHERE Granja.id_granja = ?
            AND TipoSensores.descripcion = 'Pienso retirada'
            ORDER BY Medidas.fecha DESC LIMIT 1)";

        $stmt = $this->conn->prepare($query);
        $stmt->bindParam(1, $this->id_granja);
        $stmt->bindParam(2, $this->id_granja);
        $stmt->bindParam(3, $this->id_granja);
        $stmt->execute();

        return $stmt;
    }

    function getEstAgua(){
        $query = "SELECT TipoSensores.descripcion, Medidas.medida, Medidas.id_medidas
        FROM " . $this->table_name . "
        INNER JOIN Sensores ON Sensores.id_sensor = Medidas.sensor_id
        INNER JOIN TipoSensores ON TipoSensores.id_tipo_sensor = Sensores.tipo_sensor
        INNER JOIN Granja ON Granja.id_granja = Sensores.granja_id
        WHERE Granja.id_granja = ?
        AND TipoSensores.descripcion = 'Agua'
        ORDER BY Medidas.fecha DESC LIMIT 1";

        $stmt = $this->conn->prepare($query);
        $stmt->bindParam(1, $this->id_granja);
        $stmt->execute();
```

```
    return $stmt;
}

function getEstCalefaccion(){
    $query = "SELECT TipoSensores.descripcion, Medidas.medida, Medidas.id_medidas
    FROM " . $this->table_name . "
    INNER JOIN Sensores ON Sensores.id_sensor = Medidas.sensor_id
    INNER JOIN TipoSensores ON TipoSensores.id_tipo_sensor = Sensores.tipo_sensor
    INNER JOIN Granja ON Granja.id_granja = Sensores.granja_id
    WHERE Granja.id_granja = ?
    AND TipoSensores.descripcion LIKE 'Calefacción y Aire Acondicionado'
    ORDER BY Medidas.fecha DESC LIMIT 1";

    $stmt = $this->conn->prepare($query);
    $stmt->bindParam(1, $this->id_granja);
    $stmt->execute();

    return $stmt;
}

function getEstCooling(){
    $query = "SELECT TipoSensores.descripcion, Medidas.medida, Medidas.id_medidas
    FROM " . $this->table_name . "
    INNER JOIN Sensores ON Sensores.id_sensor = Medidas.sensor_id
    INNER JOIN TipoSensores ON TipoSensores.id_tipo_sensor = Sensores.tipo_sensor
    INNER JOIN Granja ON Granja.id_granja = Sensores.granja_id
    WHERE Granja.id_granja = ?
    AND TipoSensores.descripcion LIKE '%Cooling%'
    ORDER BY Medidas.fecha DESC LIMIT 4";

    $stmt = $this->conn->prepare($query);
    $stmt->bindParam(1, $this->id_granja);
    $stmt->execute();

    return $stmt;
}

function getDatasetsMedidas(){
```

```
            $query = "SELECT AVG(Medidas.medida)
            FROM Medidas
            INNER JOIN Sensores ON Sensores.id_sensor = Medidas.sensor_id
            INNER JOIN TipoSensores ON TipoSensores.id_tipo_sensor = Sensores.tipo_sensor
            INNER JOIN Granja ON Granja.id_granja = Sensores.granja_id
            WHERE Granja.id_granja = :id_granja
            AND TipoSensores.descripcion = :descripcion
            AND Medidas.fecha LIKE :fecha";

            $stmt = $this->conn->prepare($query);
            $stmt->bindParam(":id_granja", $this->id_granja);
            $stmt->bindParam(":descripcion", $this->sensor);
            $stmt->bindParam(":fecha", $this->fecha);
            $stmt->execute();
            $row = $stmt->fetch(PDO::FETCH_ASSOC);

            $this->medida = $row['AVG(Medidas.medida)'];
        }
    }
?>
```