UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

ESCUELA POLITÉCNICA SUPERIOR DE ELCHE

GRADO EN INGENIERÍA INFORMÁTICA EN TECNOLOGÍAS DE LA INFORMACIÓN

“Desarrollo de comunicación entre microcontrolador y servidor para la programación de riego vía aplicación web”

TRABAJO FIN DE GRADO

Septiembre - 2020

Autor: Lluís Martínez Botella

Directores: Otoniel Mario López Granado

Miguel Onofre Martínez Rach

RESUMEN

Aunque en su mayoría la población dedicada al sector agrario esta poco actualizada al tratarse de gente mayor, el campo agrícola necesita actualizarse con el paso del tiempo para mejorar constantemente tanto en la producción como en calidad de vida por parte de los trabajadores. Sin necesitar una preparación previa, se pretende lograr generar una herramienta que este a disposición de estos profesionales para simplificar la gestión de sus tierras mediante la automatización del riego vía web.

Tras una instalación sobre el terreno semejante a los actuales programadores de riego, esta aplicación les permitirá realizar las mismas funciones desde la comodidad de su hogar, pudiendo así aumentar la capacidad de los elementos físicos y disminuir la cantidad de veces que necesitaran desplazarse físicamente al terreno, con el ahorro de tiempo y energía que supone.

Para ello se empleara un microcontrolador encargado de activar todos los elementos del riego automatizado mediante llamadas al servidor con un script en python que también formateara los datos para que el programa instalado en el microcontrolador sea capaz de entenderlo. Estos datos los recibirá de una aplicación web que devolverá una lista de elementos según la identificación proporcionada por el usuario y los datos que haya introducido.

Palabras clave: Aplicación web, arduino, microcontrolador, MVC

ÍNDICE

1. INTRODUCCIÓN

La domótica está creciendo cada vez más en nuestros hogares, pero no tiene por qué quedarse ahí. La comodidad producida por controlar todos los aspectos de una casa o un negocio desde tu mismo móvil de manera instantánea pulsando un simple botón se puede aplicar literalmente a todos los aspectos de la vida, en todas las aplicaciones del día a día que son rutinarias y por ende se pueden automatizar para así liberar de esta carga de trabajo a las personas que se atreven a aprender cómo utilizarlas.

La idea del proyecto es aplicar estas tecnologías también en el campo agrario, centrado en la gestión y automatización del riego de la forma más sencilla para el usuario mediante una aplicación web donde podrá seleccionar a qué horas quiere que se active su sistema de riego, pudiendo hacerlo desde su misma casa en cualquier momento y reprogramando lo que sea necesario sin tener que desplazarse físicamente al lugar donde haya sido establecido. Aunque para el mantenimiento de las instalaciones no se pueda ayudar mucho actualmente, una vez se haya instalado el producto aportará a los usuarios un mayor grado de tranquilidad y control sobre sus tierras, dando la oportunidad de monitorizar cómo va el riego o de cancelarlo en caso de lluvias, y controlar por ejemplo con qué fertilizantes dispone para el riego y todos los componentes que se puedan modificar con el paso del tiempo de un modo tan sencillo como acceder a su panel y modificando el campo en cuestión.

Mediante un microcontrolador conectado a modo de ordenador central se activarán las salidas necesarias del depósito, cubas u otros elementos físicos. Para ello un microprocesador accederá a la web con las credenciales asignadas al usuario para almacenar los datos establecidos en la base de datos, los transformará de la forma correspondiente para que el microcontrolador las comprenda y este pondrá en funcionamiento el riego en el momento indicado.

1.1. Sistemas de riego

Existen varios métodos de riego que se adaptan según las condiciones del terreno o las preferencias del agricultor, pero actualmente todos comparten la característica de estar automatizados con un horario programado. Sin contar

preferencias personales, siempre se puede encontrar la opción más apropiada según las condiciones climáticas donde se pretende instalar el sistema, las dimensiones de la parcela, el tipo de suelo y la calidad del agua.

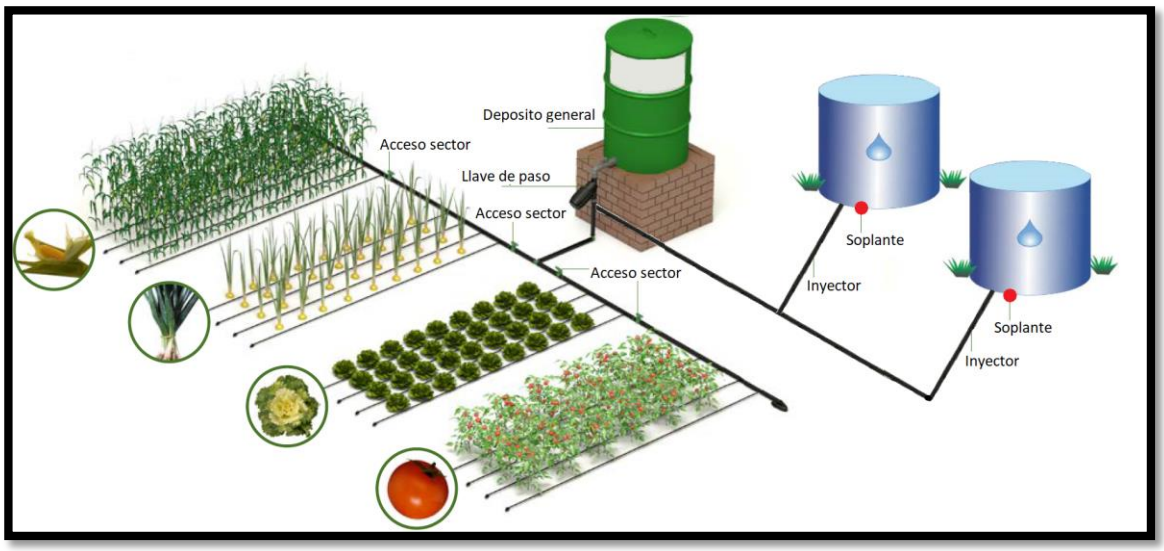

Figura 1. Componentes de un sistema de riego convencional.

Un modo muy común es el riego por goteo o riego localizado [1] que consta de un grupo de bombeo para suministrar el agua, un sistema de abonado independiente, un sistema de filtrado y todo el sistema de tuberías que se encargará de hacer llegar el agua al lugar indicado, además de filtrar el terreno según zonas definidas para generar sectores diferenciados. Las ventajas de este tipo de riego son varias, como por ejemplo un consumo más óptimo del agua, ya que suministra el agua por goteros constantemente en lugar de inundar el terreno, evitando así la evaporación del agua durante el día. Se adapta a terrenos irregulares y evita el crecimiento de malas hierbas en zonas que no se vean afectadas por el goteo. Y por último, el aspecto más importante es la ya existente automatización del sistema para el suministro de agua y de nutrientes, el cual se pretende mejorar en el desarrollo de este proyecto.

Figura 2. Cubas de riego. Sistema de abonado.

1.2. Microcontroladores MCU

Un microcontrolador [2] es básicamente un ordenador en miniatura. Tienen la capacidad de cómputo necesaria para desempeñar la tarea para la que haya sido programado, normalmente en C o ensamblador, para ser posteriormente traducido a hexadecimal, que es el lenguaje en el cual trabaja. Al igual que un ordenador convencional, dispone de una CPU encargada del funcionamiento del programa instalado en el mismo, una memoria ROM, una memoria RAM y una serie de entradas y salidas con las que se comunican con el exterior, y pueden ser de distintos tamaños y capacidades. Mediante la gestión de las entradas y salidas se mandan las señales necesarias para activar los componentes conectados o inician el proceso para que se activen varios, además de poder recibir datos sobre el estado del sistema.

Las aplicaciones de los microcontroladores van desde controlar la luz de una casa o el ratón del ordenador, batidoras convencionales, coches o sistemas de control de temperatura. Cada componente electrónico dispone de uno, por pequeño que sea.

1.2.1. Arduino

Arduino es una empresa de software y hardware libre fabricante de placas destinadas a la creación de dispositivos digitales, bajo la licencia Creative Commons [3], lo que permite modificar el código y añadir sus productos a otros proyectos con la finalidad de emplearlo adecuadamente sin repercusiones legales. Ofrece una gran cantidad de proyectos educativos para empezar desde

cero y una aún mayor comunidad con la que poder discutir sobre proyectos [4], ideas o problemas generados durante el desarrollo en sus productos, además de una serie de software de desarrollo que facilita trabajar con ello [5]. Es una herramienta muy útil para la experimentación y posterior utilización en proyectos más grandes.

Las características principales de los productos de Arduino [6] son su flexibilidad y lo económicos que son, las facilidades que ofrecen para su programación, la opción de multiplataforma y que tanto el Software como el Hardware son de código libre. Disponen de una gran cantidad de placas con características específicas preinstaladas que se pueden ajustar de un modo más óptimo a ciertos proyectos y reduce la curva de aprendizaje en el desarrollo gracias al uso de una versión reducida de C++ para su programación. Al fin y al cabo está considerada como una plataforma educativa.

1.3. Microprocesadores CPU

Figura 3. Microcontrolador integrado en una placa madre.

Se denomina microprocesador al circuito eléctrico formado por una gran cantidad de transistores que se integran en un chip. Los microprocesadores permiten el desarrollo de distintas funciones en un ordenador [7]. El microprocesador de un ordenador se compara al cerebro en un cuerpo humano, ya que es capaz de ejecutar desde aplicaciones simples hasta el sistema operativo del ordenador. Se encarga de realizar todas las operaciones lógicas necesarias para el desarrollo de las funciones deseadas mediante una serie de instrucciones en código binario transmitido mediante buses de datos.

1.4. Aplicaciones web

Las aplicaciones web ofrecen la funcionalidad de una aplicación de escritorio convencional, con la versatilidad de poder acceder a ella sin instalación y desde cualquier dispositivo con acceso a internet y un navegador (con o sin interfaz). Permiten consultar y gestionar datos, comunicarse con otras personas, redactar, aprender, comprar y vender… Pero también puede dar acceso a información sensible a gente que no debería tenerla, como contraseñas o datos personales. Esa misma es la única desventaja del uso de aplicaciones web, que se intenta subsanar mediante el uso de protocolos orientados a la encriptación y la seguridad al transmitir dicha información.

Por eso mismo, y aun teniendo un posible defecto tan grande, las aplicaciones web han ganado terreno en los últimos años, haciéndolas una de las opciones más atractivas por su accesibilidad, escalabilidad, mantenimiento y sencillez.

1.4.1. MVC (Modelo Vista Controlador)

Este patrón de arquitectura del software es un formato muy popular empleado para el desarrollo de aplicaciones web. Está orientado a la programación orientada a objetos (POO) y consta de tres componentes principales indicados en su propio nombre [8].

- Los modelos representan una plantilla que transforma todas las tablas de la base de datos en clases PHP y sus columnas en atributos, facilitando su uso posterior en los controladores y permite realizar modificaciones y consultas básicas al heredar de la clase Model. Con el uso de un mapeo objeto-relación ORM [9] podemos realizar las tareas de obtención de datos y persistencia de los mismos, además de poder crear métodos más complejos o añadir cierto nivel de lógica en las acciones que se deseen realizar. Evita tener que comunicarse directamente con la base de datos ya que las funciones habituales se pueden conseguir con la llamada consecutiva a varios métodos genéricos, facilitando en gran medida que futuros desarrolladores vean paso a paso los

parámetros de las llamadas realizadas como las cláusulas where y sus condiciones, obteniendo como resultado un elemento o lista de elementos del objeto en cuestión.

- El controlador se comunica con los modelos para recibir la información que necesita transmitir a la vista, evitando que esta sea consciente de la lógica de negocio de los modelos. Un usuario realizará una llamada al controlador mediante una ruta específica que tendrá asignada un controlador y un método, y tras recopilar todos los datos necesarios devolverá al usuario la vista solicitada completa sin necesidad de que acceda a la base de datos o el resultado de la función llamada, ya que existe la posibilidad de devolver directamente a la pantalla de error si fuese necesario. Para ello, el método llamado devolverá la vista pertinente con todos los parámetros necesarios para completarla, como listas de elementos según un filtro o textos específicos como el identificador del usuario que debe aparecer en la cabecera de página o en secciones concretas.

- La vista es el elemento que recibe el usuario. Normalmente contiene código HTML, CSS y si es necesario JS. Pueden combinarse varias vistas distintas antes de ser renderizadas, rellenando todas las variables que sean necesarias. Las vistas en el servidor están basadas en el código HTML que forman la estructura y elementos PHP que esperan recibir los datos correspondientes a cada llamada, pudiendo tratarse de datos específicos del usuario o genéricos según otros atributos como el idioma o preferencias del usuario.

1.4.1.1. Ciclo de vida de un MVC

Representado en la figura 4 quedan representados gráficamente los pasos que se siguen de manera ordenada en el ciclo de vida de un MVC para

traducir la petición de un usuario en la vista deseada desde el inicio hasta el fin de la petición y todos los agentes que intervienen.

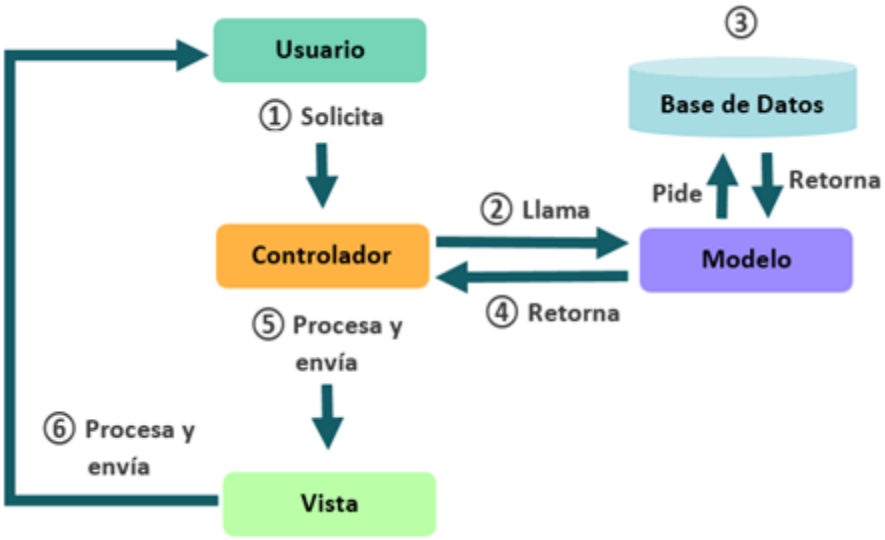

Figura 4. Ciclo de vida del MVC [8].

El usuario solicita acceder a la información del servidor mediante una ruta (p.e. "www.sitio.es/info"), petición que llega al controlador correspondiente según la configuración que se le haya asignado a la ruta "info" y recopila la información mediante llamadas a los modelos necesarios. Estos se comunican con la base de datos, reciben todos los registros que se correspondan con la petición establecida en el controlador, se devuelven en forma de elemento o lista de elementos del objeto Model definido y en caso de ser necesario el controlador trata los datos recibidos antes de enviarlos a la vista, que generará el código HTML necesario sustituyendo las etiquetas PHP con los datos recibidos del controlador. Para finalizar el ciclo, la vista será devuelta al usuario, que podrá leer e interactuar con la misma.

1.4.1.2. Ventajas y desventajas

La desventaja principal de los MVC es la alta curva de aprendizaje comparando con otros patrones, ya que existe una mayor cantidad de archivos a mantener y existen más capas en el sistema, pero facilita el manejo de errores, la escalabilidad de la aplicación, generar distintas representaciones de datos según las diferentes vistas o permisos del usuario sin necesidad de generar otro documento individual mediante el uso de condiciones a las vistas y sobre todo,

evita tener que emplear código HTML en conjunto con PHP crudo, sino que permite el uso de etiquetas que hace el código más visual y manejable.

1.5. Alcance del proyecto

En este proyecto se pretenden utilizar los servicios web tanto para el usuario como para combinarlos con el uso del microcontrolador para conseguir una comunicación sencilla mediante una aplicación REST que el microcontrolador pueda utilizar para acceder a los datos almacenados en el servidor y pueda encender y apagar los sensores encargados de realizar la tarea asignada. Para ello es necesario tener un acceso a internet desde el terreno también, considerando la cobertura disponible de la zona, aunque en este proyecto utilizaremos una conexión WiFi estándar desde nuestra propia casa, por lo que la implementación de un sistema diseñado para su utilización en zonas de poca cobertura no entraría dentro del objetivo principal.

Se almacenan las credenciales del usuario, así como los datos del horario y cuándo encender y apagar cada pin en la memoria principal para que el microcontrolador pueda acceder a ellas y realizar las consultas fácilmente, manteniendo actualizada lo máximo posible dicha información mediante llamadas consecutivas al servidor.

En la parte web se tienen en cuenta las medidas estándar de seguridad sin ahondar demasiado en ellas, manteniendo un control de la integridad de la base de datos mediante los formularios y controles antes de añadir dicha información a la misma. Lo más importante es la usabilidad antes que el diseño, por eso se centra más en la posición y comportamiento de los distintos botones y formularios que en el diseño de una interfaz atractiva. Se ofrecen todas las acciones necesarias para que el usuario disponga de un control total de los datos susceptibles de ser modificados por el mismo, con restricciones en sistemas considerados como configuración de la aplicación necesarios para el correcto y óptimo funcionamiento de la misma.

1.6. Material utilizado

Este proyecto consta de dos piezas básicas, la aplicación web y una placa arduino Yun Rev2.

La aplicación web, desarrollada completamente en Laravel y accesible desde el dominio “http://matrix.umh.es:8502/tfgriego”, será la herramienta de la que dispondrán los usuarios para crear o modificar un horario ya implementado en sus respectivas parcelas. Dispondrán de un desplegable donde seleccionar la parcela que deseen ver o modificar, así como los horarios programados para la misma. En esta lista aparecerá una parcela por cada unidad que se haya “adquirido” de nuestro producto y haya sido dada de alta previamente, que actualmente sirve solamente como demostración del funcionamiento deseado.

El Arduino Yun Rev2 será a la vez el microcontrolador y el microprocesador instalado en la parcela que funcionará como lo que ahora suele ser el programador de riego [10] de las parcelas. La elección de esta placa se basa en que ya viene integrado con un Shield encargado de la comunicación con el microprocesador, que dispone de una versión reducida de Linux llamada Linino, el cual permitirá utilizar ciertas funcionalidades básicas de un sistema operativo, además de disponer de un adaptador WiFi. Esto facilita la comunicación, ya que es gracias al SO que se podrá consultar la web sin necesidad de un navegador. En una tarjeta SD instalada en la placa se almacena el código en Python que se encargará de hacer la llamada a la URL apropiada y almacenará los resultados tratados de manera que el microcontrolador los comprenda posteriormente. Una vez almacenados, el microcontrolador se mantendrá comprobando la hora hasta que coincida con la de un riego programado, empezando a leer los datos y encender y apagar las salidas en el orden especificado.

Al no disponer de los elementos necesarios para implementar este proyecto en una parcela real, el correcto funcionamiento de la aplicación se comprobará gracias a una protoboard [11] con una serie de luces led que simbolizan la activación de cada uno de los activables. Conectando cada pin a una luz de distinto color según el tipo que sea, por ejemplo los sectores representados por una luz azul, se realizará un símil con el uso real y el correcto control correspondiente a los datos introducidos en la base de datos.

2. PROYECTO

2.1. Microcontrolador, Arduino Yun Rev2

La placa utilizada se puede encontrar en la tienda oficial de Arduino [12], con un precio alrededor de los 50€. Una vez adquirida la placa, una protoboard y todos los elementos necesarios para realizar las pruebas de funcionamiento, pasamos a la instalación de todos los componentes necesarios.

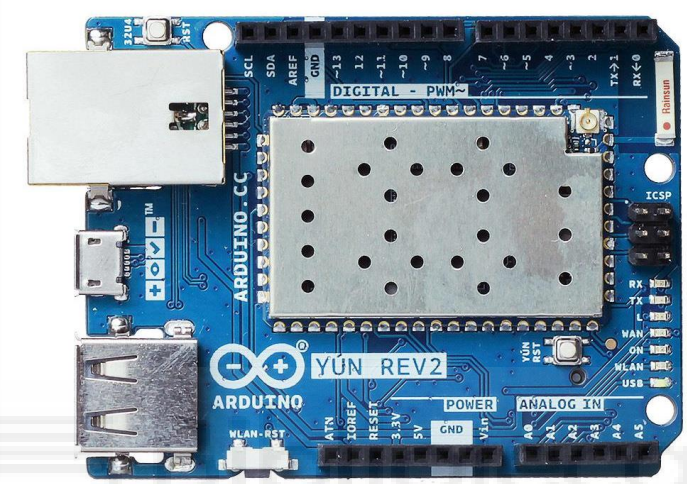

Figura 5. Placa arduino Yun Rev2.

No se ha generado ningún presupuesto al considerarse un entorno de pruebas, en el que el único gasto sería el de los componentes anteriormente mencionados, siendo los gastos poco destacables para el ámbito del proyecto en el caso de una implementación física a mayor escala.

2.1.1. Instalación

Descargamos la imagen actualizada de OpenWrt de la página oficial y metemos la imagen en la tarjeta SD formateada que se planea utilizar, asegurándonos de que está formateada en formato FAT. No debe haber problemas cuando se hace esto con la tarjeta recién comprada, pero es recomendable asegurarse. Conectamos la placa por ethernet a nuestra misma red para configurar la red WiFi por primera vez y se conecta con el navegador a la ip que le haya asignado el router, simplemente accediendo al panel de control y buscando esa misma información. Con el nombre predeterminado “arduino”, identificarlo es una tarea sencilla.

Se actualiza la imagen pulsando sobre el botón rojo al final de la pantalla y se selecciona la red con la que se pretenda trabajar, escribiendo las credenciales de la red y ya se podrá reiniciar el Yun para que se conecte automáticamente por WiFi sin necesidad de estar conectado por cable. Mientras no se cambie de red ya no será necesario volver a realizar esta configuración. Aproximadamente tarda un minuto en conectarse a la red cada vez que se enciende de nuevo para poder acceder a la misma, programando el microcontrolador desde el editor online de Arduino o con la aplicación de escritorio. En este caso utilizaremos la herramienta online, que detectará automáticamente la placa siempre y cuando el ordenador esté conectado a la misma red.

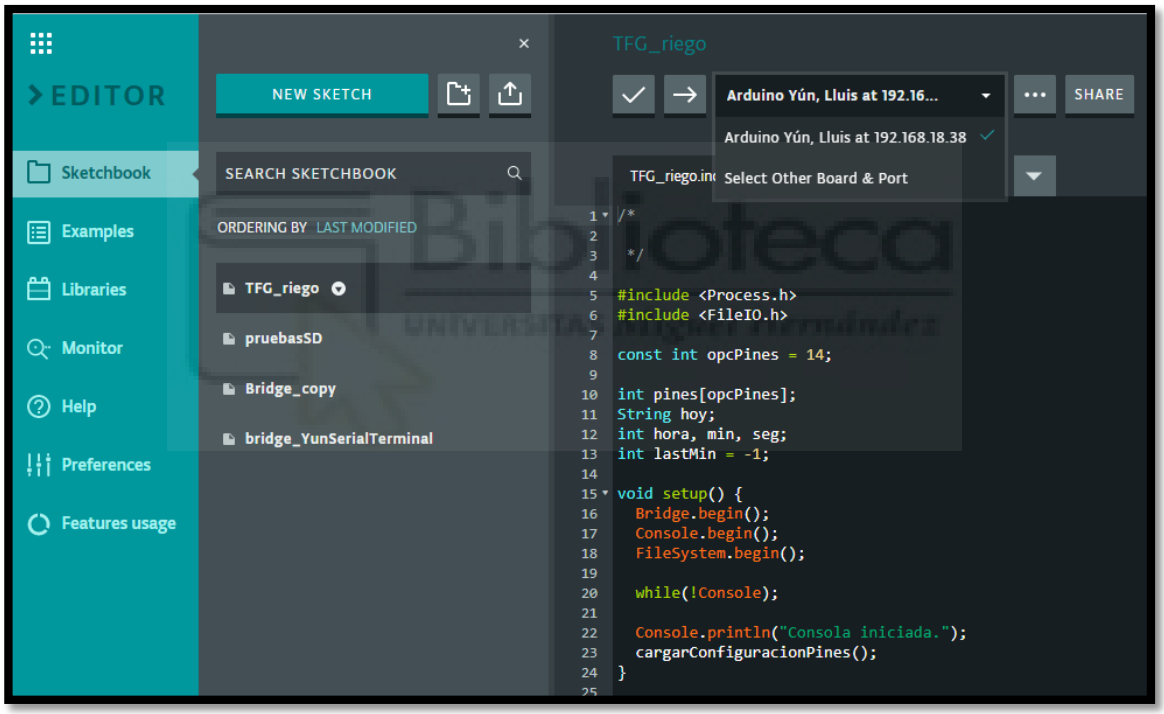

Figura 6. Herramienta online editor de Arduino con función setup.

Todos los pasos anteriormente mencionados y otras opciones para configurar del mismo modo la placa para en su primer uso se pueden encontrar en la documentación oficial de arduino [13].

2.1.2. Microprocesador Linino

Esta placa funciona con un procesador Atheros AR9331 [14] con 64 MB de DDR2 y el sistema Linino preinstalado en su memoria flash de 16 MB. Esta

memoria tiene un número limitado de escrituras, por lo que evitaremos siempre que sea posible modificar los elementos del mismo y se priorizará el uso de la tarjeta micro SD instalada anteriormente. Conectando la tarjeta SD se almacenarán todos los datos y programas necesarios para acceder a estos. También dispone de un puerto USB que nos ofrece las mismas posibilidades, pero en este caso es innecesario.

Para estandarizar el funcionamiento del Yun, dentro de la ruta "/mnt/sda1" se crea una carpeta llamada "arduino", recomendación del tutorial de instalación de la propia marca para acceder de manera más sencilla desde el microcontrolador a la información mediante el puente preinstalado en la placa [15]. Ahí mismo cargamos el script encargado de realizar la consulta al servidor mediante una llamada al sistema con el comando "curl", añadiendo los datos del usuario y el identificador de la parcela asignada para realizar la llamada al servicio REST en formato POST para mantener la mayor seguridad posible sobre estos datos. El resultado de esta llamada será un JSON, formateado en el servidor antes de transferirlo y almacenado en un archivo txt al lanzar el resultado de la llamada en el archivo con el nombre especificado mediante el comando ">" que creara o modificara el archivo indicado como segundo parámetro con la información recibida tras lanzar el comando "curl". Posteriormente en el mismo script los datos recibidos serán tratados para generar archivos individuales por cada horario en la lista.

Dada la menor capacidad del microcontrolador, tanto ROM como RAM, la información es reducida para que se pueda leer línea a línea sin comprometer el correcto control de la información ni cargar demasiada información a la vez. Para ello se genera un cubo de tres dimensiones, donde cada una de ellas será; la hora a la que hay que activar cualquier elemento, si se debe encender o apagar y la lista de elementos según identificador que deben actuar en consecuencia [Figura 7]. Los elementos que deban activarse una sola vez durante un cierto rango de tiempo se añadirán directamente a la lista de horas, una para el inicio y otra para el fin, en la lista de encendidos y apagados respectivamente, pero en el caso de que el campo periodicidad sea distinto de cero significa que se tienen que tratar más a fondo. A partir de la hora de inicio y de manera sucesiva, se asigna una entrada de este elemento a la lista de encendido y apagado de

manera intercalada a razón del periodo indicado hasta que se llegue a la hora de fin de riego.

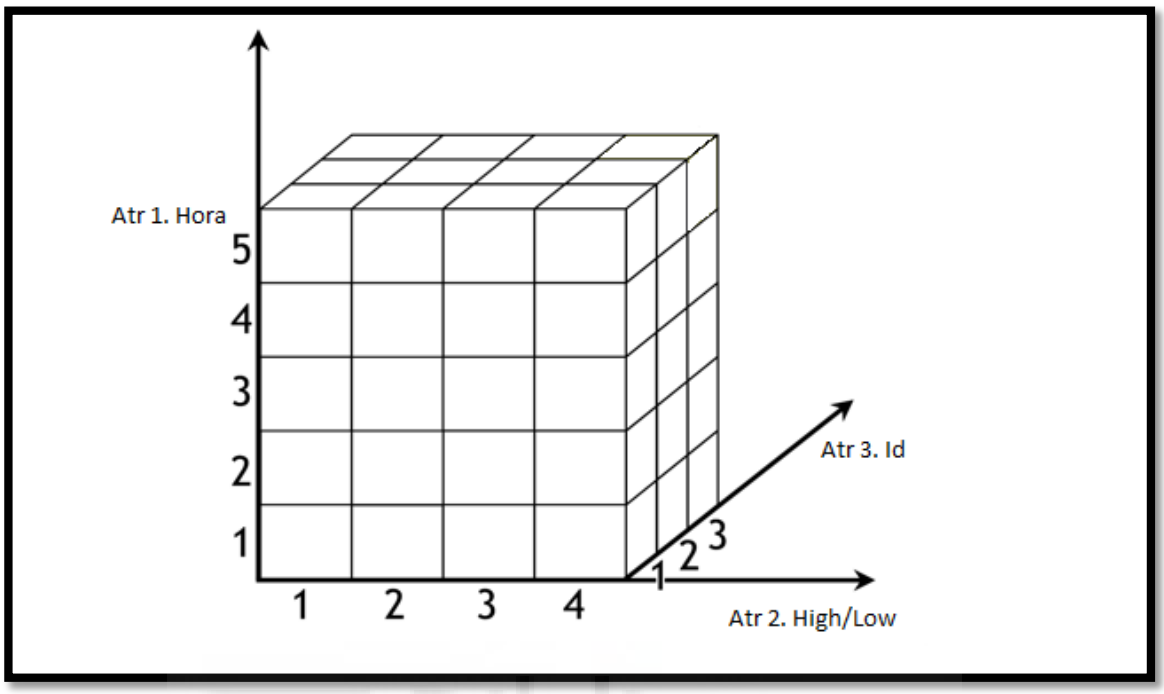

Figura 7.Cubo generado a partir de un horario.

Por último, independientemente de estos factores, se añadirá una señal de apagado para todos los elementos a la hora estipulada de fin de riego, que en el caso de los elementos periódicos asegura el apagado del mismo, aunque por asignación dé la casualidad de que vuelva a encenderse justo a la misma hora del fin.

Una vez recopilados todos los datos, el cubo se transformará en una serie de identificadores, precedidos por la hora y el valor 1 en caso de encendido y el 0 en caso de apagado. La forma que se le dan a los datos en este paso es crucial a la hora de programar el microcontrolador, porque leerá los datos directamente de este archivo y no tendrá otro modo de acceder a ellos. El formato que tendrá que leer el microcontrolador será de la forma “HORA-(“0”/”1”):id:id:id:...:id” considerando que cada id se traducirá al pin correspondiente según la asignación en el archivo de configuración que se menciona a continuación.

2.1.3. Microcontrolador ATmega32u4

Este microcontrolador [16] dispone de 20 pines digitales y 12 analógicos, pero para nuestro caso contaremos con que son 32 pines ya que los utilizaremos todos de modo digital, cantidad más que suficiente para la aplicación del proyecto, que necesitará reservar alrededor de 10 como máximo.

Cada uno de estos pines se asigna a un identificador de los activables relacionados con la parcela a la que pertenece de manera manual, a modo de mantenimiento para asegurar que se corresponden a la realidad en un archivo de configuración llamado pines.conf que será común para los distintos horarios, ya que representa una asignación para con los elementos de la parcela. Al comprobar la fecha actual se busca una carpeta que coincida con el día de hoy. Si no se detecta ninguna, simplemente se queda en a la espera hasta que se dé la ocurrencia y se prepare para la lectura. El microcontrolador supone que el script ha guardado los horarios de forma ordenada según la hora, por lo que leerá línea a línea el archivo que se encuentra dentro de la carpeta correspondiente al día en cuestión, cargando en memoria solamente la lista por horas una a una, realizará un bucle hasta quedarse sin elementos a modificar y esperando hasta la siguiente hora. Asignará el valor encontrado (0 o 1) a todos los pines que encuentre posteriormente según el separador asignado ":", hasta encontrar la próxima hora y mantenerse a la espera o simplemente al encontrar el último carácter del fichero, acabando el riego y volviendo a realizar la espera en búsqueda de otro día programado. Para asegurar que esté lo más actualizado posible se repetirá la llamada al servidor para comprobar si se ha realizado algún cambio en el horario previsto para dar la posibilidad de cancelarlo o de realizar modificaciones a última hora.

2.1.4. Librería Bridge

Como se ha mencionado anteriormente, la comunicación se realiza mediante un puente preinstalado en el Yun, que gracias a esta librería permite que el microcontrolador acceda a la información del microprocesador y viceversa. Esta función ocupa los pines Rx y Tx como cualquier otra comunicación con el microcontrolador, pero de un modo más simplificado gracias a la librería Bridge, la cual pone a disposición una serie de clases con sus

respectivos métodos para lograr el comportamiento deseado. En la figura 8 podemos observar la estructura y el sentido de las comunicaciones entre los distintos componentes de la placa.

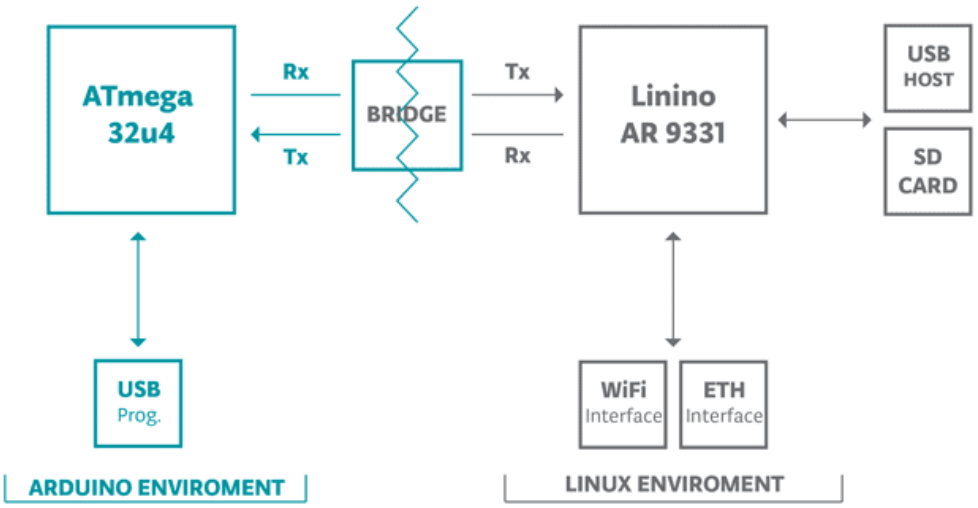

Figura 8. Esquema de comunicación y estructura de la placa Yun Rev2.

Las funciones más relevantes de la librería para el proyecto se explican a continuación.

2.1.4.1. Clase Process

Esta clase [17] permite realizar llamadas al sistema, recibir y tratar el resultado desde el microcontrolador. Declarando una variable del tipo Process e indicando con el método begin el comando que queremos lanzar. Añadimos parámetros en caso de necesitarlo con el método addParameter y lo ejecutamos con run. Esto resultará en un String con lo que el sistema haya devuelto almacenado en el objeto Process, que se puede leer tras realizar la llamada con el método readString.

Con este método el microcontrolador podrá recibir sencillamente datos como la hora actual con el comando “date”, ver si existe alguna carpeta de horarios con “ls” o lanzar de manera periódica el script de python, aprovechando el elemento loop del microcontrolador para mantener esta información lo más actualizada posible a la hora de iniciar el riego ya que este bucle se ejecutara

constantemente mientras siga funcionando el dispositivo. Estas son las funciones que se utilizaran en este proyecto, pero se podrán utilizar todos los comandos disponibles en la versión de Linino del microprocesador ya que su funcionamiento simula las llamadas al sistema que realiza cualquier desarrollador en línea de comandos al conectarse directamente a la placa.

2.1.4.2. Clase FileIO

Con esta clase [18] se pueden leer, escribir y gestionar los archivos de la tarjeta SD. Aquí es donde se empieza a utilizar la carpeta Arduino que se crea en la raíz de la tarjeta, la cual crea automáticamente un vínculo a la ruta "/mnt/sd", lo que nos permitirá estandarizar los Strings que especifiquemos en el sketch para acceder a la ruta donde los elementos deben haber sido creados. En este caso no es de mucha utilidad ya que solo se dispone de una tarjeta SD, pero la documentación recomienda crearlo para facilitar su uso.

Lo primero es iniciar en el setup de Arduino el sistema de archivos con el método begin, tras lo que se podrán emplear las clases y métodos necesarios para abrir archivos, recorrerlos o comprobar si existen. Ofrece también otras funciones como abrir archivos consecutivamente con openNextFile, útil para posibles modificaciones en las que se pretenda almacenar todos los archivos en la misma carpeta para eliminar rutas innecesarias, y crear o borrar directorios sin necesidad de realizar llamadas al sistema, pero en el planteamiento de este proyecto no son necesarias.

2.1.4.3. Clase Console

Por último, la clase Console [19] permite añadir mensajes a modo de depuración para comprobar el correcto funcionamiento de la aplicación o detectar en qué punto de la misma recorre un flujo de datos distinto al esperado. Al igual que el gestor de archivos, es necesario iniciarlo en la setup de Arduino, e incluso pausar el funcionamiento de la aplicación hasta que detecte que se ha realizado una conexión por telnet. Estos datos se utilizarían como registros de actividad y de error en el momento que esté totalmente operativo, dando la

oportunidad de hacer un seguimiento de su estado en caso de registrar dichos mensajes y utilizarlos para detectar algún error en el funcionamiento.

Para conectarse a la consola se debe hacer desde el Linino. Primero se debe conectar por SSH al microprocesador mediante Putty o por línea de comandos. Después, empleando el comando telnet se indica la ruta localhost para conectarse a la interfaz de la consola ya que la conexión se realizara de modo local considerando la previa conexión al dispositivo Yun.

Figura 9. Acceso por SSH a la placa Yun desde el entorno de pruebas.

2.2. Aplicación web, Laravel

2.2.1. Interfaz de la aplicación web

Las vistas empleadas en el proyecto habilitan al usuario a utilizar todas las funciones necesarias para configurar su parcela. Dispone de todas las páginas necesarias en caso de actualizar la aplicación con un aspecto más comercial para cuando fuese necesario, pensando que en fases más avanzadas los usuarios primero deben acceder a toda la información que les fuese necesaria, como funcionamiento, precio, instalación del producto, entre otros, pero no considerándose necesarias para el ámbito de este proyecto.

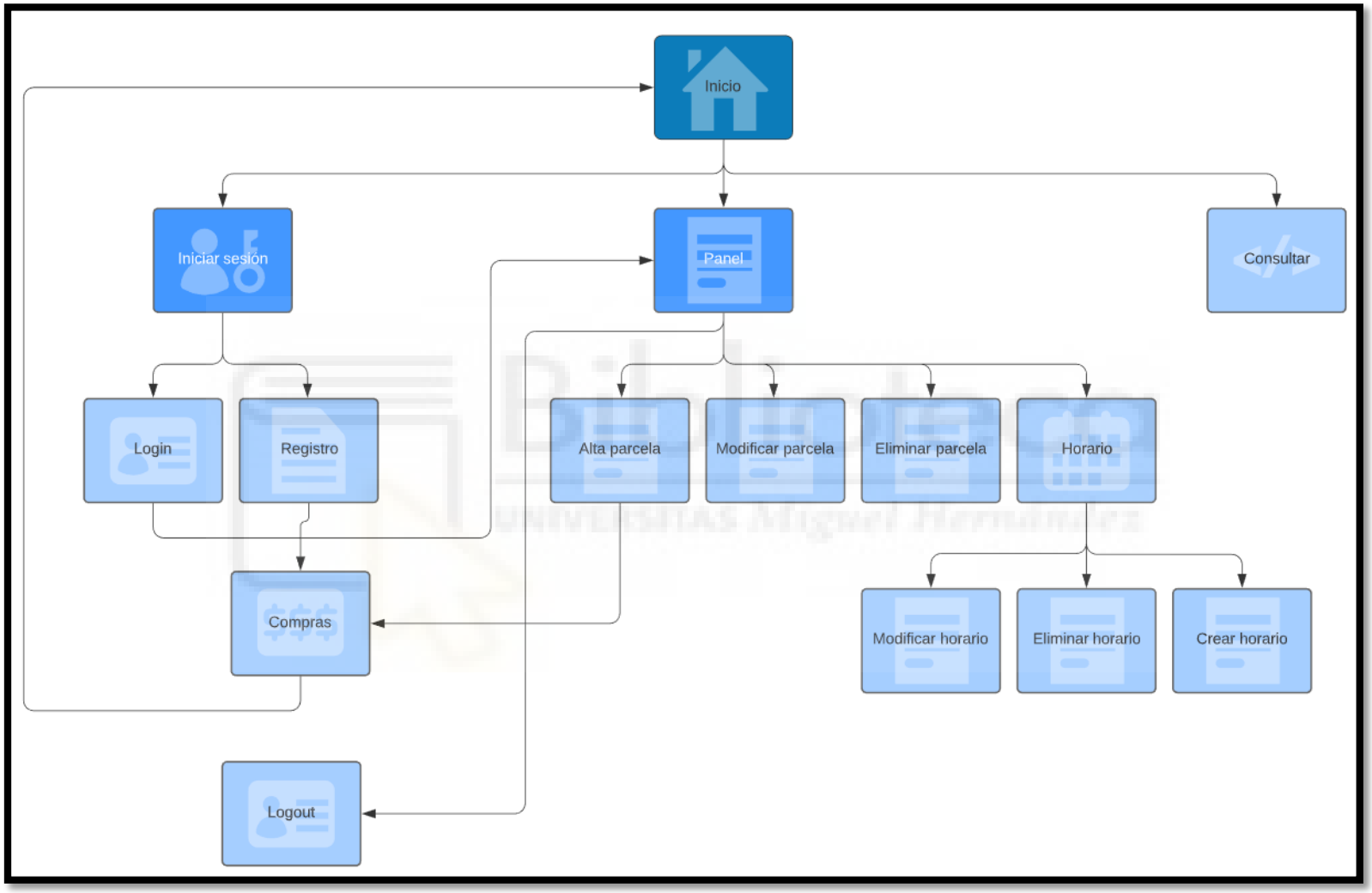

Figura 10. Mapa del sitio web.

En la Figura 10 se muestra el mapa completo del sitio web, con las rutas principales que se utilizan. Cuenta con un menú superior estándar, donde si no existe una sesión activa, dará acceso al login o a la página de registro, y una vez registrado generará un menú de accesos directos a las gestiones disponibles sobre las parcelas. Para generar horarios es necesario especificar primero la parcela a la que pertenece el mismo, así que no se añadirán estas funciones en el menú.

El login se basa en email y contraseña estándar, pero el registro incluirá un campo “clave de yun”, que será la simulación de un código de activación que el usuario recibirá al simular la compra de una licencia. Al recibir el formulario se comprobará que dicho código cumpla los requisitos para ser un código válido antes de habilitar el registro y crear al nuevo usuario

Una vez registrado se accede al panel principal, donde se dispondrá de un desplegable que muestra todas las parcelas relacionadas con el usuario, dando a elegir entre todas las direcciones de las mismas como identificador para que el usuario reconozca rápidamente la parcela con la que desea interactuar. Una vez seleccionada, en la misma vista se visualizarán los datos relativos a dicha parcela en forma de formulario para dar la opción a modificar los datos informativos de estos (los datos como número de sectores o de cubas no se podrán modificar, ya que en una parcela estos no suelen cambiar nunca y supondrían un cambio importante en la estructura de la base de datos para los elementos relacionados). Desde esta vista también se accederá a la creación y modificación de los horarios en la misma parcela seleccionada, pero para acabar con la gestión de parcelas, pasaremos a la parte del menú desplegable. En este se podrá acceder al menú para dar de alta una parcela, donde se solicitarán los datos físicos antes mencionados, que no podrán modificarse desde la aplicación al tratarse de datos más sensibles que alteran en gran medida la base de datos y la gestión de todos los elementos posteriormente, y la asignación de un Yun a la parcela para evitar que se den de alta más parcelas, suponiendo que solo se puede tener una por cada Yun que se haya adquirido. También se podrá acceder a la eliminación de las parcelas, que dispondrá del mismo desplegable que en la sección de información, y simplemente se dará de baja la parcela seleccionada,

así como todos sus elementos relacionados. La relación con la realidad física y la de la base de datos será un tema de integridad del cual deberá responsabilizarse el usuario al introducir los datos de la parcela.

En cuanto a los horarios, con la parcela seleccionada se listaran todos los asignados a esta, mostrando el día y el rango horario en el cual se seleccionó al darlo de alta. Cada uno tendrá su propio botón para la modificación, funcionando igual que la modificación de parcelas, otro de borrado, que llevara a una pantalla de confirmación con los datos mostrados individualmente para que el usuario no borre ninguno por equivocación, y por último botón en la parte superior para asignar un nuevo horario a la parcela. En este caso primero se seleccionará el inicio y el fin del mismo y se asignará como horario estándar al resto de elementos de la parcela, dando la posibilidad posteriormente a modificar estos rangos para cada sector, la asignación de abonado según las cubas disponibles y el inicio y periodicidad del soplado de las cubas en caso de estar disponibles en dicha parcela.

Por último, se dispondrá de una pantalla de compra, donde la información no ha sido generada por ser este un aspecto comercial. Simplemente tendrá un botón “Comprar” que te redirigirá a la pantalla principal con un código de activación mostrado en pantalla que será el que se utilizará en el registro, o en su defecto se asigna directamente al usuario registrado en caso de encontrar una sesión activa. La selección del Yun a asignar es aleatoria, y cuando se asigna a un usuario aparecerá como desplegable a la hora de dar de alta una nueva parcela, y en caso de no tener ninguno, no podrá darse este caso, ya que teóricamente se necesita un microcontrolador previamente instalado o al menos en proceso de instalación en la parcela para poder funcionar.

La ruta de consulta no dispone de interfaz, ya que será la ruta encargada de devolver los datos de todos los horarios a los Yuns ya instalados.

2.2.2. Servidor web

El servidor tiene instalado Laravel 5.8 con composer 1.8.5 para la gestión de dependencias. En cada uno de los apartados se indica las funciones que se realizan en cada una de las rutas relevantes para este proyecto. No difieren

mucho de las básicas que vienen instaladas por defecto en cualquier otra aplicación Laravel, pero algunas funciones simplifican la gestión de los formularios y reducen la carga de código dentro de los controladores. A continuación se explica la funcionalidad de cada una de las rutas destacables y que funciones realizan dentro de la aplicación:

2.2.2.1. /app/Http/Controllers

Cada uno de los controladores [20] se encarga de la lógica de cierta parte de la aplicación. Se dividen según los objetos relacionados con las tareas que deben realizar para mantenerlos más fácilmente y evitar agrupar todas las funciones dentro de la misma clase, generando una clase con excesivas líneas a mantener.

Disponemos del sistema básico de autenticación que genera Laravel [21] de manera automática en la carpeta “Auth”. Al lanzar el comando de instalación se generan todos los controladores y el Middleware necesario para manejar las funciones básicas de registro e inserta elementos en las vistas como la barra superior de Login y Sing In, además de todas las vistas necesarias para una gestión básica de los usuarios y las migraciones de la tabla de usuarios. Lo primero tras instalarse es eliminar la vista “home”, ruta estándar a la que redirige la aplicación en todos los casos de identificación satisfactoria, pero en este caso la aplicación debe llevar directamente a la gestión de parcelas en el panel principal. En cada uno de los controladores de autentificación generados debemos modificar el atributo redirectTo a la vista del panel en lugar de home. Esta variable está presente en todas las clases relacionadas con la autentificación, que es uno de los atributos que pueden definir dicho comportamiento. Al modificar esta variable en lugar de otras se asegura que siempre se redirige a la ruta deseada, ya que es la más priorizada por Laravel.

El único controlador que se debe modificar en cuanto a funcionalidad de los instalados por defecto es el de registro, ya que necesitamos la condición de que el Yun sea apto para su asignación. En la función de validación, la entrada del código Yun debe cumplir una nueva regla Aquired, explicada en el apartado de Rules. Tras superar la validación y crear el nuevo usuario, dicho Yun debe

asignarse al usuario recién creado para que el dueño del mismo pueda declarar su primera parcela con ese Yun y asegurar que no puede registrarse cualquier persona incondicionalmente sin simular antes la compra del producto. Una vez verificado que todos los datos son correctos, se debe modificar el método de creación de usuarios para que además de crearlo, modifique también los elementos asociados mencionados.

Para las parcelas es necesario especificar un método distinto para cada una de las funciones básicas, que son; ver el listado de las parcelas y sus datos, modificar los datos de la seleccionada, dar de alta una nueva y borrar una específica. Cada una de estas funciones tendrá su método dentro del controlador con un nombre lo suficientemente representativo como para entender qué hace cada una con tan solo leerlo. Considerando que es necesario distinguir entre tipos de llamadas HTTP, cada una de estas tendrá dos secciones donde comprobará primero el método con el que se ha realizado la llamada, con el método GET o el correspondiente en formato POST, ya que cada uno tendrá su función, pero si la llamada es GET se supone que es una petición simple para recibir la vista que ofrecerá al usuario la opción de introducir los datos necesarios para la interacción, posiblemente con algún parámetro que especifique la parcela con la que se desea interactuar.

En este caso, las funciones de visualización y modificación son conjuntas para evitar generar más vistas y que el usuario pueda moverse entre parcelas e ir modificando las mismas sin tener que volver a la vista anterior. Los campos donde se visualizan los atributos de la parcela son a su vez entradas de un formulario que tomará efecto en caso de pulsar el botón inferior de la pantalla, situado al final para que el usuario tenga que pasar por todos los campos y así intentar generar un último repaso de los datos que se van a introducir antes de su confirmación.

Para eliminar una parcela primero se cargará la misma lista que aparece a la hora de seleccionar la que se desea visualizar, pero en una vista apartada donde el título de la misma es claramente el de “Borrar parcela” para evitar confusiones. Al seleccionar una de las parcelas y presiona en borrar, el Yun que estuviese asignado pasaría a estar libre y se podría volver a utilizar en el registro

de una nueva parcela para el mismo usuario. Después se procede a eliminar la parcela y todos los datos relacionados con la misma, tanto activables como horarios. Al ser la parcela un elemento principal de la BBDD se eliminan muchos registros de la misma, ya que la mayoría dependen directamente de ellas y todas las acciones con las relaciones en este caso serán en cascada, dependiendo de la parcela para poder existir.

Para dar de alta, al igual que en el registro de usuarios, es necesario tener algún Yun en propiedad relacionado con el usuario que haya iniciado sesión. Este paso es el más crítico, ya que el dar de alta una parcela conlleva la creación de una gran cantidad de elementos relacionados que son todos los activables que físicamente se pretenden controlar. Al dar de alta hay que ser muy concreto con la cantidad de elementos de las que dispone la parcela real e introducir los datos de manera que coincida totalmente. En caso de error no se podría modificar, ya que esta función realiza varias llamadas a los modelos Activables y crea sus entradas en la base de datos según los parámetros introducidos en el formulario de creación. Estos son el número de sectores, número de cubas, la existencia de soplantes en estas, el yun que debe estar instalado o se pretenda instalar y datos menos relativos como la superficie total o la dirección. Este último parámetro es el que el usuario tendrá que utilizar para identificar qué parcela estará tratando posteriormente, ya que suponemos que la dirección es única al tener que añadir el número de la misma y no deben existir dos calles con el mismo nombre, al menos en la misma población y considerando poco probable que una misma persona posea dos parcelas con el mismo nombre de calle a no ser que sean la misma físicamente, por lo cual deberán tener un número distinto. En la figura se muestra un ejemplo de cómo comprobar el tipo de llamada que recibe el controlador y como realizar distintas acciones en cada caso. Se aprecia que al intentar declarar una parcela nueva, si la petición es GET primero busca la lista de Yuns pertenecientes al usuario actual y la utiliza para que el usuario seleccione cuál de los Yuns será el que instalará en la nueva parcela. Si esta llamada no encuentra ningún Yun disponible, el usuario tendrá que ir primero a la sección de compras, adquirir uno y volver a la página de alta. Una vez introducidos los datos, el formulario realizará una llamada PUT y pasará a realizar la otra parte del código.

```php
public function nuevaParcela(Request $request){

    if($request->isMethod('get')){
        if(Auth::user()){
            $yuns = Auth::user()->yuns;
            $disponibles = $this->yun->sinUsarPorUsuario(Auth::user());
        }
        return view('alta', ['yuns' => $disponibles]);

    }else if ($request->isMethod('put')){
```

Figura 11. Ejemplo del uso distintivo según petición HTTP.

Antes de introducir los datos en la base de dato se realizan unos filtros de validación para evitar que se sobrepasen los límites propios de la placa Yun, como por ejemplo la consideración de que habrá como máximo diez elementos activables, cuatro llaves de paso, una para cada sector, dos cubas, dos soplantes en el caso de existir, el inyector de las cubas y la llave de paso general. Los dos últimos se dan de alta automáticamente en cualquier caso, y el resto según los datos especificados en el formulario, con los límites preestablecidos en la validación.

Como la lista de parcelas del usuario se utiliza en varias vistas también disponemos de una función para recuperar directamente una lista formateada como array para poder transmitirla a la vista y recorrer todos los elementos, mostrando en cada caso la información que proceda según la identificación proporcionada.

El controlador encargado de los horarios también realiza las mismas funciones que en las parcelas, además de encargarse de transmitir los datos al microcontrolador. Para visualizar la lista de horarios existentes se utiliza una lista de vistas predefinidas como plantillas para cada horario relacionado con la parcela seleccionada, rellenando los datos y añadiendo para cada uno un botón encargado de borrar o modificar dicho horario. Para identificarlos se muestra la fecha y las horas en las que empieza y acaba el riego definido, suponiendo que no habrán dos iguales, ya que en ese caso solo se aplicaría el último en declararse al solaparse en los datos del Yun.

Para dar de alta primero se pide el día y horas de inicio y fin del riego, que una vez seleccionados se tomarán como predeterminados para el resto de elementos que cargaran justo debajo, pudiendo dividir la franja horaria posteriormente para cada sector y el inicio de riego de las cubas o dejarlo como está, ahorrando al menos la modificación de la mitad de estos elementos y facilitando al usuario rellenar todos los campos solicitados. Cada elemento se identifica con su propio id al cargar la vista para que cuando el usuario haya incluido todos los datos del modo que lo necesite, el controlador sepa a qué elemento se refiere cada grupo de datos, generando a parte del propio horario en la tabla correspondiente, todos los horarios de activables necesarios según la parcela que se esté tratando. Para cada elemento asignará el inicio del riego y la duración o periodicidad del activable, según tenga que encenderse y apagarse a dos horas concretas o si tiene que hacerlo más de una vez durante un tiempo específico. La duración correspondiente a cada horario se calcula restando la hora de inicio a la hora de fin y será almacenada en minutos.

El borrado llevará a una pantalla de confirmación donde se deberá confirmar el borrado del horario, mostrando solamente los datos del elemento que se pretende borrar, dando la opción al usuario de revisar si es el horario deseado y la posibilidad de rectificar en caso de ser necesario.

La modificación de los horarios será semejante a la de las parcelas, pero en una ruta individual identificando la parcela en la cual se ha generado dicho horario y su identificador para posteriormente volver a la vista de horarios completa. Esto es debido a que no se puede reutilizar la vista que los muestra porque no está basado en un desplegable, sino en varias plantillas por la necesidad de mostrar varios elementos del horario para que el usuario pueda identificarlos fácilmente y ver todos los elementos al mismo tiempo. Aparecerá un formulario muy semejante al de creación de horarios mostrando directamente los datos que se especificaron en su creación, que el usuario podrá modificar a su gusto y devolverlos al servidor para que el cambio se haga efectivo.

La última función de este será la de devolver una lista en formato JSON ya que estos datos se pretenden emplear en otras aplicaciones, por lo que estandarizar su formato lo máximo posible ayudará en su manejo una vez fuera

del servidor. Estos datos se devolverán siempre y cuando se indique en la petición POST los datos acreditativos del usuario y la parcela de la cual se desean obtener los horarios. Utilizando la librería Auth se comprueba rápidamente que el usuario exista y llama a la relación existente entre el id de la parcela y sus horarios.

Para acabar con las funciones de los controladores, el de Compras se encarga de mostrar la pantalla de compra y de actualizar la base de datos cuando se simula la compra del producto. Este controlador tiene poca funcionalidad ahora mismo, pero en un futuro estará listo para añadir nuevas interacciones, ya que la posibilidad de que el negocio requiera una lógica específica para las compras es muy grande. Por ahora simplemente simula la compra seleccionando el código de activación de un Yun al azar y devolviéndolo para que el usuario lo utilice en su registro, o si bien el usuario ya está registrado, se le asigne la propiedad al mismo automáticamente.

2.2.2.2. /app/Models

Para cada tabla en la base de datos se crea un modelo que permitirá acceder de un modo más simple a los datos almacenados y recibir las relaciones con un solo comando. Es posible especificar métodos para recibir o modificar cada una de las columnas de la tabla con Accessors y Mutators [22], pero no será necesario ya que Eloquent [23], el ORM utilizado en Laravel por defecto, genera de modo automático una lista de atributos con el mismo nombre que las columnas de la tabla y una vez declarado un objeto que herede de la clase Model se puede acceder a ellos mediante propiedades del mismo objeto, devolviendo los datos almacenados en cada uno de sus atributos. También existe la posibilidad de crear métodos genéricos para simplificar el código en los controladores, como por ejemplo la asignación del Yun a un usuario dado, o comprobar si un Yun existe o no. Esto evita tener que incluir algunas de las librerías varias veces y genera un código más limpio al eliminar líneas o evitar sobrecargar las clases.

```php
public static function acquired($codigo){
    $yun = Yun::find(1)->where('codigo_activacion', '=', $codigo)->first();

    return ( !($yun === null) && !(Yun::hasOwner($yun))) ? true : false;
}

public function sinUsarPorUsuario(User $user){
    return Yun::find(1)->where('id_user', '=', $user->id)->where("activo", '=', 0)->get();
}
```

Figura 12. Funciones específicas para el control de Yuns en el modelo.

Los modelos también se encargan de generar las relaciones [24] con otros modelos para acceder a ellas como atributos cualesquiera. Como recomendación la documentación de Laravel muestra la creación de métodos con el nombre de la tabla a la cual se pretende acceder, devolviendo una llamada a los métodos disponibles para recibirlas, que dependiendo de las cardinalidad de cada una de las relaciones será un método u otro. En caso de que una tabla no necesite consultar dicha relación o no se tenga intención de hacerlo no es necesario declararla, pero si recomendable en caso de que la aplicación crezca, ya que es más complicado comprobar cuáles de estos métodos faltan que simplemente generar todas las relaciones.

```php
/*Relaciones Eloquent*/
public function user(){
    return $this->belongsTo('App\User', 'id_user');
}

public function parcela(){
    return $this->hasOne('App\Models\Parcela', 'id_yun');
}
```

Figura 13. Ejemplo de la declaración de relaciones en Model.

Los métodos a los que se deben llamar para recibir esta información, aunque se tienen a disposición otros para relaciones más específicas, son los métodos belongsTo y hasOne o hasMany dependiendo de la cardinalidad, con la ruta del modelo de la relación correspondiente. En caso de modificar la clave principal de la tabla en la base de datos se debe especificar en un tercer parámetro, pero en este caso todos los identificadores se han nombrado como el estándar de Laravel, con la forma de “nombreTabla_id”

2.2.2.3. /app/Rules

Las reglas [25] son una clase encargada de la validación de formularios principalmente. Permite generar la lógica de una regla más compleja que la de ser texto, o que un número esté dentro de un rango y permite realizar todas las comprobaciones que se deseen para acabar devolviendo un booleano que indique si se acepta o no se acepta la información introducida en los formularios. En caso del no, también permite especificar un mensaje de error que se mostrará al usuario con la etiqueta @error que mencionaremos más tarde.

La utilidad reside en la posibilidad de ejecutar comprobaciones en la base de datos para, a partir de los datos introducidos en un parámetro del formulario poder asegurar que cumple unas condiciones específicas que no comprometan la base de datos ni causen ningún efecto adverso en la aplicación. En la documentación oficial de Laravel sobre reglas se dispone un listado de todas las reglas predefinidas en la aplicación.

2.2.2.4. /config

Como su propio nombre indica, aquí se almacenará toda la configuración relativa a la aplicación [26], como las credenciales de la base de datos o el nombre de la misma. Permite simplificar llamadas en el código especificando alias para ciertos servicios, como Auth o BD, que nos permiten acceder a toda la librería sin especificar la ruta completa en cada uno de los controladores. Dispone de muchas más opciones, pero no se entra en detalle por la poca utilidad que se le puede dar a un entorno de pruebas como es el caso de este proyecto. Consideramos como aspecto destacable la conexión a la base de datos, donde se pueden especificar todos los datos para acceder a esta.

2.2.2.5. /database

La gestión de la base de datos desde la propia aplicación ayuda al desarrollo en un entorno de pruebas, además de mantener una integridad absoluta en caso de necesitar generarla o restaurarla. Las migraciones de Laravel [27] permiten describir tabla a tabla, columna a columna, los detalles necesarios y mantenerlos en una ruta individual, pudiendo crear los documentos

manualmente o mediante el comando "make:migration" y ejecutarlos con el comando "migrate".

Las clases Migration constan de dos métodos, up y down, que se lanzan al crear y al eliminar las migraciones respectivamente. Para generar la tabla lo único que hay que hacer es generar un plano especificando cada una de las columnas necesarias con nombre, tipo y todos los atributos que fuesen necesarios llamando a la lista de métodos predeterminados que ofrece la clase Blueprint. Por último, para las relaciones se debe hacer en dos llamadas distintas, declarando primero la columna que almacena la relación y después especificando de nuevo la columna y el nombre de la tabla a la que queremos que haga referencia, así como su comportamiento en caso de modificación.

```
$table->bigInteger('id_yun')->unsigned()->unique();
$table->foreign('id_yun')
      ->references('id')->on('yuns')
      ->onDelete('cascade')->onUpdate('cascade');
```

Figura 14. Declaración de relaciones en las migraciones.

Las clases Seeders [28] poblarán la base de datos con una serie de elementos especificados manualmente que también se llamarán de manera automática al encontrarse dentro del método run. Aquí se puede añadir una lista en forma de array con todos los elementos marcados como obligatorios para cada una de las tablas, ya que en caso contrario lanzará un error al intentar ejecutarlos. Esto se debe especificar en la clase del modelo con el atributo fillable que indica cuáles de las columnas se pueden asignar masivamente. Una vez se tengan preparados ambos conjuntos de clases ya se puede generar una versión inicial de la base de datos en cualquier momento y reiniciarla en caso de comprometer su integridad por cualquier motivo. En este caso se deja un usuario y una parcela de ejemplo con todos sus activables para no tener que registrarse cada vez que reiniciemos. Ejecutando el comando "php artisan migrate:fresh --seed" se llama a las clases generadas y crea de nuevo las tablas y los registros especificados. Eliminando el último parámetro se ejecutaran solo las migraciones.

2.2.2.6. /resources/views

Mediante el motor de renderizado Blade [29] que utiliza Laravel por defecto se consiguen generar las vistas necesarias que recibirá el usuario. Todo el código de las vistas está encapsulado lo máximo posible para dar una mayor velocidad a la actualización de las mismas, con la mayor cantidad de plantillas posibles para simplificar las vistas principales, cargando vistas secundarias con bucles foreach según los datos recogidos del servidor. Los elementos repetibles, como la información de sectores y horarios están almacenados en la carpeta de vistas bajo la ruta "/includes" y otra específica para las vistas del horario para dar un orden más eficaz a la hora de encontrar las vistas necesarias. Estas son las que los controladores deben devolver al final de su función y pueden contener tanto texto plano como elaboraciones complejas gracias a los datos recibidos en cada una, dando la opción a mostrarse o no en caso de que cierta variable exista en el ámbito concreto, ya que por ejemplo se puede utilizar la misma vista mostrando una parte de la misma, y al recibir algún dato extra desde la petición se añadirán más secciones como podemos observar a continuacion [Figura 15.a, 15.b].

Figura 15.a. Interfaz en la primera llamada al panel principal.

Figura 15.b. Interfaz tras seleccionar una parcela en el menú superior.

Esto permite mostrar ciertas secciones solo si se da el caso de que existan al encontrarse en la base de datos o al haber completado otro campo anteriormente, o de no mostrarse por haber accedido a la vista por un camino distinto al esperado. Un ejemplo de este caso es cuando se da de alta un horario. La primera llamada solo muestra día, inicio y fin, pero luego se vuelve a llamar a la misma ruta, encontrando los datos anteriormente mencionados, por lo que pasa a cargar otros elementos de la página.

Al ser el cuerpo de todas las páginas de la misma forma, es decir, todas necesitan la barra superior donde acceder al menú de login o el de gestión de parcelas, y en un futuro el footer de la aplicación también deberá tener un aspecto más comercial, todas las vistas heredarán del mismo diseño almacenado en la ruta "/layouts/app.blade.php". Esta vista contiene una etiqueta @yield, que determina el nombre de la etiqueta @section que espera encontrar en las vistas que pretendan heredar de ella. Entonces, para generar una vista

completa que herede de la principal se debe declarar una etiqueta @extends con el nombre de la principal y a continuación, considerando la posibilidad de añadir texto entre medias, se define qué código deberá incluirse en la etiqueta @yield especificado entre una etiqueta @section y otra @endsection. Aunque sea posible, no es conveniente incluir código fuera de estos componentes, ya que la aplicación estandariza tanto la cabecera como el pie de página, así que cualquier código fuera de dicha sección se mostrará al final del documento, es decir, por debajo del footer.

Los bucles se emplean con la etiqueta @foreach, donde su funcionamiento es totalmente igual al de cualquier otro lenguaje. Todo el código que se encuentre entre esta etiqueta se repetirá tantas veces como elementos se encuentren en el array especificado como argumento, pudiendo así añadir una vista para cada elemento con los datos concretos de cada uno. Para diferenciarse es necesario añadir un elemento identificador que se recoge en el servidor a la hora de asignar los datos a cada uno, generando otra lista de elementos con el mismo nombre y añadiendo "[]" al final. De este modo, al mandar el formulario se recibirán varios arrays con los datos distintivos para cada uno de los elementos insertados.

También es destacable el uso de la etiquet @guest para evitar el acceso sin autorización a las vistas, ya que en caso de no detectar una sesión iniciada previamente, la aplicación devuelve otra vista predefinida en la que se pide al usuario que se registre. El menú superior seguirá disponible para que el usuario recuerde registrarse o identificarse y acceda a dichos formularios sin tener que salir de la pantalla que haya devuelto el aviso. Este será en todo caso el menú de login, ya que en la misma plantilla app se utiliza esta etiqueta para decidir si se muestra el login o el menú relativo a las parcelas del usuario. También ofrece la opción de mostrar según roles, pero se ha decidido no especificar funcionalidades fuera del ámbito del usuario.

Las etiquetas @error indican la posibilidad de recibir como respuesta al intento de mandar un formulario un mensaje de error al no cumplir las reglas de validación, mostrando donde se haya especificado dicha etiqueta el mensaje de error indicado en la regla del servidor que no se haya cumplido.

2.2.2.7. /routes/web.php

El enrutamiento [29] se gestiona desde este documento. Se debe especificar el nombre de la URL, el controlador que lo gestiona y qué método se encargará de gestionar la llamada dentro de dicho controlador, un nombre para facilitar su llamada en las vistas y que método de petición HTTP utilizara. Simplificando código, en lugar de especificar un solo método, se utiliza la misma ruta con distinto método request para llamar al mismo método con distintos tipos de formularios, ya que en algunos casos el formulario puede enviarse de varios modos o más de una vez para obtener resultados distintos y reutilizar algunas vistas y métodos. Es importante asegurarse de que todas las funcionalidades sean accesibles y estén definidas en este documento, ya que la aplicación no podrá traducir la URL del usuario en un funcionamiento lógico a no ser que pueda acceder a dicha lógica desde este archivo.

2.3. Base de datos

Lo primero a la hora de desarrollar la base de datos es pensar en que va a necesitar la aplicación, qué elementos deben existir y con cuales se relacionan. Mencionaremos a continuación las especificaciones necesarias para cada una de las tablas que se ha decidido crear y la razón de estas decisiones, considerando que se intenta simular un sistema de riego por goteo.

2.3.1. Tablas de la base de datos

2.3.1.1. Tabla users

Como en cualquier otro proyecto, los usuarios deben almacenarse en algún lugar para gestionar las tareas de identificación y protección de datos, tanto propios como internos. Además de las columnas genéricas necesarias para almacenar e identificar a los usuarios, como son el nombre, correo o contraseña, que ya se incluyen al instalar la autentificación predeterminada de Laravel, otros campos destacables son la dirección del usuario para en caso de comercializar el producto, poder enviarlo a su propia casa o utilizarlo en otro tipo de gestiones. En este caso el usuario no deberá almacenar ninguna relación, ya que puede disponer de varias parcelas al igual que de varios Yuns para cada una de ellas.

2.3.1.2. Tabla parcelas

En esta tabla se almacenan meramente datos simbólicos que ayuden al usuario a identificar de qué parcela se trata al seleccionarla en la lista, además del identificador de usuario para saber a quién pertenece. También almacenará el identificador del Yun que se le haya asignado, ya que en la misma parcela sólo puede haber instalado uno. En caso de necesitar más de un controlador en aplicaciones futuras dadas las limitaciones de la placa empleada, se puede modificar la estructura y añadir los pines necesarios para seguir manteniendo la misma estructura

2.3.1.3. Tabla yuns

Como ya se ha indicado anteriormente, los Yuns serán los encargados de recordar quién es su propietario ya que cada uno solo podrá pertenecer a una persona. Esta tabla necesita mantener toda la información para asegurar que el Yun no está siendo utilizado equivocadamente, por ejemplo por futuras versiones que cambien algún aspecto del funcionamiento o configuración, la fecha en la que se realizó la compra del mismo para asegurar que no pueda utilizarse antes de comprarlo, un código de verificación para utilizarlo en el registro de usuarios y un indicador de que ya se ha instalado correctamente y esta funcional. Muchos de estos aspectos se deben desarrollar en futuras versiones de la aplicación, pero se declaran ahora aun con su funcionalidad reducida para facilitar el futuro mantenimiento.

2.3.1.4. Tabla activables

Representan todos los elementos dentro de una parcela que son susceptibles de hacer alguna tarea. Por ejemplo, cada sector necesitará que se abra la llave de paso para que llegue el agua, o la cuba tendrá que echar el abono que contenga durante cierto tiempo. Eso implica una dependencia, por ejemplo, por parte de los sectores que tendrán que tener implícita la activación de la válvula general correspondiente, que será otro activable derivado cuyo horario dependerá totalmente del conjunto de horas de todos los sectores de la parcela. Para gestionar el comportamiento de estos y todas las relaciones con

activables derivados existe la relación con tipo_activable, que indicará dentro de una parcela qué tarea se le debe asignar exactamente.

2.3.1.5. Tabla tipo_activables

En una parcela existen varios tipos de elementos, y como acabamos de mencionar, cada uno tendrá tareas distintas y por lo tanto deben comportarse acorde a su finalidad. Al existir una cantidad finita de elementos bien definidos, se decide crear esta tabla para almacenar el nombre de los mismos ya que no supondrá una carga mucho mayor y permite actualizar la lista de elementos en caso de querer añadir nuevos tipos u otras funcionalidades al tener que actualizar la misma. Por ahora sólo almacenan una descripción de lo que representa cada uno de los activables, pero la falta de esta tabla podría complicar la escalabilidad en un futuro al no permitir lo anteriormente mencionado.

2.3.1.6. Tabla sectores

Aunque para el usuario esta parte es meramente informativa, ya que almacenará los datos relativos al contenido físico real y actual del sector (este puede variar según temporadas, meses, o simplemente que el usuario modifique lo que esté cultivando por placer) pero no aporta ninguna funcionalidad a la aplicación ya que de eso se debe encargar el activable asociado. Se crea una entidad débil para asegurar que existe una lista de propiedades de dicho activable y que esté relacionado directamente con este.

2.3.1.7. Tabla cubas

Las cubas mantendrán un comportamiento similar al de los sectores en cuanto a su utilidad para almacenar datos relativos al activable además de tener relación con el soplante relacionado en caso de existir. La única diferencia será una doble relación con la tabla activables, una como entidad débil y otra para especificar su soplante.

Durante la creación de la parcela se generan las cubas indicadas por el usuario, se crean y se asignan a estas también un elemento soplante en caso de haberse indicado por parte del usuario y se relacionan entre si para mantener las relaciones de funcionamiento posteriormente.

2.3.1.8. Tabla horarios

Un horario representa el día en el cual se pretenda regar, el inicio y el fin. La duración depende más de lo especificado en los activables, ya que en algún momento puntual puede no haber ningún elemento en funcionamiento pero no haber acabado el riego, haciendo al campo fin no tan crítico dentro de la aplicación, pero sí conveniente tenerlo por mera información. No debería darse el caso en el que el fin supere los límites de la acción real, y como se ha mencionado anteriormente es una información muy útil para agilizar la inclusión de datos en los formularios a la base de datos. Además, teniendo en cuenta posibles modificaciones en la aplicación, podría ser un buen indicador de situación crítica para dejar de considerar movimientos tras la hora especificada.

2.3.1.9. Tabla activable_horarios

Puede considerarse la tabla más importante de toda la aplicación, ya que hace posible la comunicación con el Yun de la manera más rápida posible. Teniendo en cuenta que una lista de elementos con estos atributos es lo que el Yun utilizara para saber cuál es su cometido, deberá contener todos los datos de los activables para que se enciendan o apaguen en su momento o repitan la acción cada cierta cantidad de tiempo. Pensando en que claramente un horario contendrá más de un activable, pero esos mismos activables también tendrán varios horarios para cada uno de los días deseados, es necesaria una relación de muchos a muchos entre ambas tablas, donde además de los identificadores foráneos de las otras tablas necesitaremos el inicio esperado del activable, la duración estimada y en caso de ser repetitivo, una duración en minutos del tiempo que tendrá que mantenerse encendido o apagado. Esta relación genera la necesidad de esta tabla, que en conjunto contendrá toda la información necesaria para formatear la configuración del funcionamiento del microcontrolador.

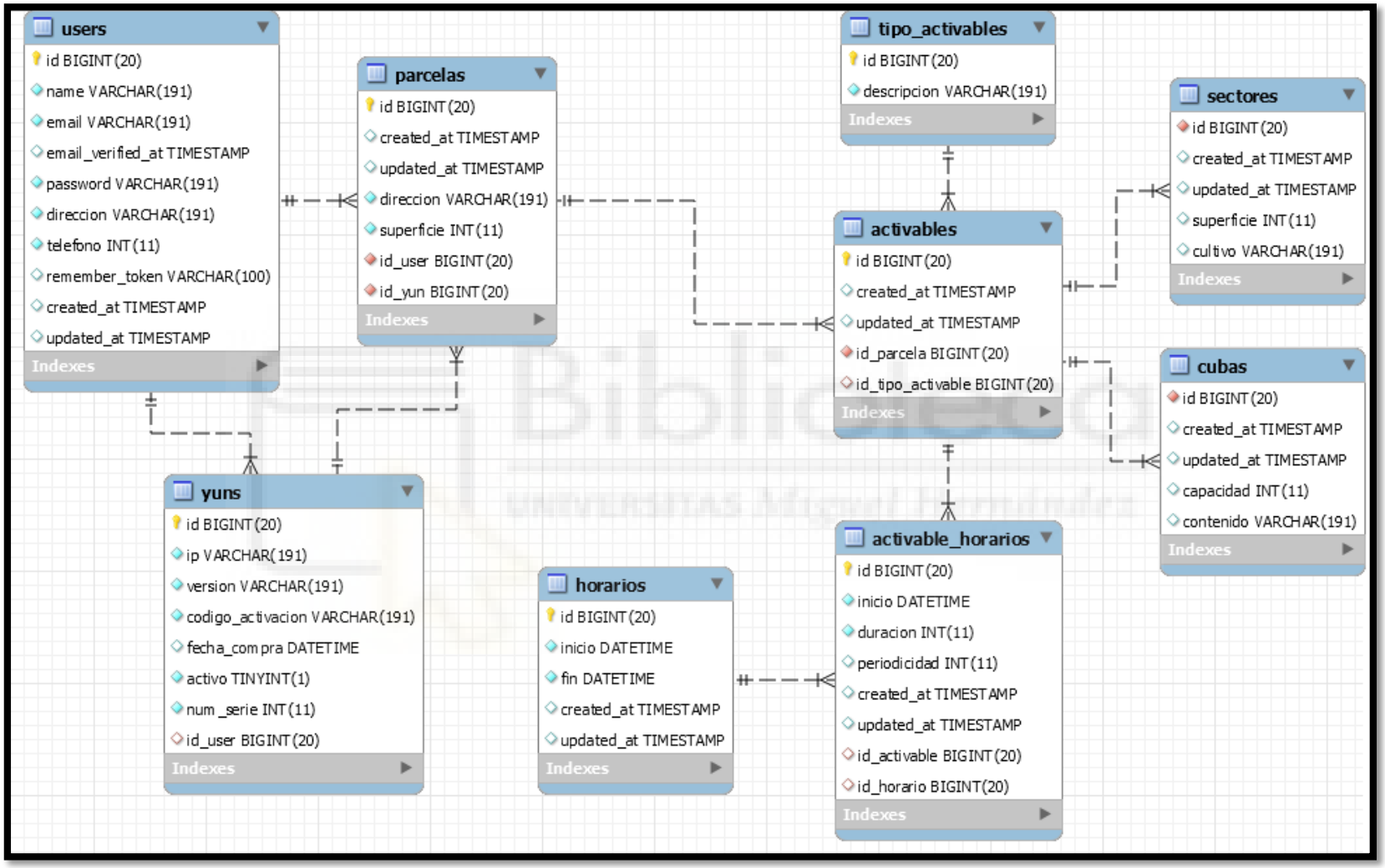

Figura 16. Esquema final de la base de datos.

2.3.2. Aspectos a tener en cuenta sobre la BBDD

Los campos especificados en la tabla usuarios son un simple ejemplo para ver cómo podría ser útil esta información en algún momento de la vida de la aplicación y de cómo modificar la plantilla al ser instalada automáticamente con ciertos elementos que podrían no ser necesarios en un futuro, pero no se pretende ahondar en el uso de usuarios más que en el hecho de necesitar una identidad como en cualquier otra página y la necesidad de agrupar más de una parcela. Algo parecido pasa con las parcelas, que en un ambiente de pruebas no es necesario más que un par de campos para facilitar el desarrollo, pero en una aplicación real serán necesarios otros identificadores para evitar así la creación masiva de registros en la base de datos y posiblemente serán necesarios más datos identificativos que corroboren la propiedad del terreno por parte del usuario.

En el caso de la tabla yuns también son elementos meramente orientativos sin ninguna validez actualmente, ya que se deberá desarrollar la forma de los mismos más detenidamente si se fuese a utilizar en un ambiente más profesional. Los códigos de activación ahora mismo son cuatro dígitos tecleados al azar en la población automática de la base de datos para disponer de cierta cantera de productos con los que probar las funciones de compra y activación, pero estos códigos deberán cumplir ciertas medidas de seguridad en un futuro mediante la encriptación y uso de aplicaciones para generar estas claves aleatoriamente [30].

Por último, la tabla que almacena la relación entre horarios y activables es la más susceptible de ser modificada en un futuro, ya que cualquier cambio en la forma de controlar los elementos obligará a una modificación en el comportamiento de la placa. Esto significa que cualquier cambio en la tabla podría entorpecer los algoritmos generados en el microprocesador para su almacenamiento aunque los datos recibidos siempre tengan el formato JSON.

3. CONCLUSIONES

En general los objetivos del proyecto han sido totalmente satisfactorios, ya que se ha conseguido una aplicación funcional para el usuario que el microcontrolador traducirá en los datos que necesita para el correcto funcionamiento y es capaz de controlar variables como la hora a tiempo real. En caso de conectar elementos ligeramente más complejos sería posible generar una prueba más completa, pero dado el tamaño de la placa el encendido de leds se considera suficiente.

Mediante el uso de la aplicación web el usuario será capaz de ver, modificar y eliminar sus parcelas y horarios de un modo óptimo e intuitivo para que la dificultad de uso sea mínima. En caso de posible fallo de integridad, Laravel ofrece todos los elementos necesarios para mostrar un mensaje que el usuario entienda y pueda volver a intentarlo, por lo que el usuario no necesita esforzarse en pensar qué ha hecho mal.

Con el microcontrolador el acceso a internet era el aspecto más preocupante por la necesidad de identificar al usuario automáticamente desde la misma placa sin necesidad de introducir las credenciales constantemente, pero gracias al método de consulta de horarios se puede comprobar la existencia de los argumentos especificados en la petición y si alguno no estuviese presente o no se encontraran, simplemente no se devolverá nada. Esto demostraría que es un acceso no deseado o un fallo de configuración al instalar la placa, por lo que se podría considerar un aspecto de mantenimiento. También se han empleado llamadas al sistema que, aun siendo más rudimentarias, evitan tener que instalar más librerías en la limitada memoria de la placa, lo que podría resultar en una necesidad de ampliación posterior y una dependencia de librerías innecesaria a cambio del esfuerzo extra que supone realizar estas tareas tan sencillas de este modo.

Aunque funcional, se podría añadir otros métodos de control como el borrado de horarios anteriores a la fecha actual en la memoria del microcontrolador, ya que estarán obsoletos. Esta parte podría no ser tan relativa por el hecho de que, considerando el uso actual de los programadores de riego, en lugar de por días la programación podría ser semanal y cíclica, ya que

normalmente los riegos suelen ser así, y en lugar de programar los días específicos de riego con su fecha, las opciones serían modificar los días de la semana y que se repitan sucesivamente con el orden semanal establecido, pudiendo cambiar un día concreto antes de que se active esa semana y después restaurar los horarios anteriores en caso de ser un caso específico o simplemente preservar las nuevas especificaciones.

Otra perspectiva futura es la creación de funcionalidades orientadas a la gestión como administrador, o posiblemente dar ciertas libertades al usuario de configurar más opciones del microcontrolador a parte de los horarios, siempre y cuando dichos elementos no comprometan el correcto funcionamiento de ninguno de los componentes. Esto se debe a la correlación actual entre los pines de la placa y el elemento físico relacionado, que en este caso se maneja con un archivo de configuración en la memoria del microcontrolador. Al no tener interfaz que modifique estos datos ni el microcontrolador estar programado para esperar ningún cambio, lo hace un aspecto crítico en el correcto funcionamiento de la aplicación.

4. BIBLIOGRAFÍA

[1] Riego por goteo, Novagric, https://www.novagric.com/es/riego/sistemas-de-riego/riego-por-goteo.

[2] Descubrearduino.com, microcontrolador, https://descubrearduino.com/microcontrolador/.

[3] Creative Commons, https://creativecommons.org/licenses/?lang=es_ES.

[4] Foro oficial Arduino, https://forum.arduino.cc/.

[5] Software de arduino, https://www.arduino.cc/en/main/software.

[6] Descubriendo Arduino, https://descubrearduino.com/.

[7] Definición de microprocesador, https://definicion.de/microprocesador/.

[8] MVC, ¿Qué es?, http://rodrigogr.com/blog/modelo-vista-controlador/.

[9] ¿Que es un ORM?, https://programarfacil.com/blog/que-es-un-orm/.

[10] Programadores de riego, Novagric, https://www.novagric.com/es/riego/materiales-de-riego/programadores-de-riego

[11] Protoboard-Breadboard, qué es y cómo se usa, https://hetpro-store.com/TUTORIALES/protoboard-breadboard/.

[12] Tienda oficial de Arduino, https://store.arduino.cc/arduino-yun-rev-2

[13] Getting Started with the Arduino Yún, https://www.arduino.cc/en/Guide/ArduinoYun.

[14] Microprocesador Atheros datasheet, https://www.openhacks.com/uploadsproductos/ar9331_datasheet.pdf.

[15] Bridge Library for Yún devices, https://www.arduino.cc/en/Reference/YunBridgeLibrary.

[16] Microcontrolador atmega32u4 datasheet, http://ww1.microchip.com/downloads/en/devicedoc/atmel-7766-8-bit-avr-atmega16u4-32u4_datasheet.pdf.

[17] Clase Process, https://www.arduino.cc/en/Reference/YunProcessConstructor.

[18] Clase FileIO, https://www.arduino.cc/en/Reference/YunFileIOConstructor.

[19] Clase Console, https://www.arduino.cc/en/Reference/YunConsoleConstructor.

[20] Documentación oficial de Laravel, Controllers, https://laravel.com/docs/5.8/controllers.

[21] Documentación oficial de Laravel, Authentication, https://laravel.com/docs/5.8/authentication.

[22] Documentación oficial de Laravel, Mutators, https://laravel.com/docs/5.8/eloquent-mutators.

[23] Documentación oficial de Laravel, Eloquent: Getting Started, https://laravel.com/docs/5.8/eloquent.

[24] Documentación oficial de Laravel, Eloquent:Relationships, https://laravel.com/docs/5.8/eloquent-relationships.

[24] Documentación oficial de Laravel, Validation, https://laravel.com/docs/5.8/validation.

[25] Documentación oficial de Laravel, Configuration, https://laravel.com/docs/5.8/configuration.

[26] Documentación oficial de Laravel, Database:Migrations, https://laravel.com/docs/5.8/migrations.

[27] Documentación oficial de Laravel, Database:Seeding, https://laravel.com/docs/5.8/seeding.

[28] Documentación oficial de Laravel, Blade Templates, https://laravel.com/docs/5.8/blade.

[29] Documentación oficial de Laravel, Routing, https://laravel.com/docs/5.8/routing.

[30] Generate License Key (GENLICKEY), https://www.ibm.com/support/knowledgecenter/ssw_ibm_i_72/cl/genlickey.htm.