Proyecto Fin de Carrera
Ingeniería de Telecomunicación

DESARROLLO DE APLICACIÓN MÓVIL MULTIPLATAFORMA PARA PROGRAMACIÓN DE RIEGO

Autor: José Antonio Zaiño Rodríguez
Tutor: José Manuel Fornés Rumbao

Dpto. de Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Proyecto Fin de Carrera
Ingeniería de Telecomunicación

DESARROLLO DE APLICACIÓN MÓVIL MULTIPLATAFORMA PARA PROGRAMACIÓN DE RIEGO

Autor:
José Antonio Zaiño Rodríguez

Tutor:
José Manuel Fornés Rumbao

Dpto. de Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2020

Proyecto Fin de Carrera: DESARROLLO DE APLICACIÓN MÓVIL MULTIPLATAFORMA PARA PROGRAMACIÓN DE RIEGO

Autor: José Antonio Zaiño Rodríguez

Tutor: José Manuel Fornés Rumbao

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2020

El Secretario del Tribunal

A mi familia.

A todas las personas que me han hecho llegar aquí.

AGRADECIMIENTOS

Tras un gran esfuerzo por parte de muchas personas, puedo concluir satisfactoriamente mi carrera con este documento. No habría podido llegar aquí sin el apoyo de todas ellas. Por este motivo, me gustaría agradecer en este apartado a cada una de las personas que han hecho posible que esté redactando esto.

A cada miembro de mi familia, especialmente a mi padre José Antonio y a mi madre Ana.

A mi hermano Jesús.

A José María Baeza, por sus conocimientos, su tiempo y aportaciones a la realización hardware y software en este proyecto.

A mis amigos y a mis compañeros de carrera, mi familia en este viaje.

A mi pareja, por su ayuda, paciencia y apoyo.

RESUMEN

Hoy en día, la evolución e inclusión de Internet se ha convertido en una necesidad de primer orden, estando presente desde los procesos más simples hasta los más complejos, los cuales se pueden desarrollar en muchas ocasiones de manera telemática.

Junto a esta creciente evolución, está estrechamente relacionado el desarrollo de la tecnología de los smartphones, los cuales son, junto con sus aplicaciones, las herramientas que nos permiten monitorizar y manejar múltiples aplicaciones y herramientas de Internet.

El área de extensión de las aplicaciones móviles es incalculable, siendo hoy día muy relevante en la creación de entretenimiento para el usuario (videojuegos, redes sociales…) y en el control o automatización de sistemas que, sin la existencia de estas aplicaciones, requerirían de la presencia del factor humano para su funcionamiento.

Uno de los campos con más auge de la automatización y el control telemático es el agrario, facilitando y simplificando muchos de los procesos, como los sistemas de riego, los cuales tiempo atrás requerían de una gran cantidad de recursos humanos, materiales, económicos… para poder llevarse a cabo de manera correcta.

Con el desarrollo de este proyecto se pretende realizar la automatización de un gestor de riego, de forma autónoma, a través del desarrollo de una aplicación. Esta aplicación permitirá la gestión de distintos campos, cultivos y riegos de un usuario, pudiendo añadir tantos como desee tener, planificando los riegos o encendiendo los riegos de manera manual.

Dicho desarrollo se realizará con la tecnología React Native haciendo la aplicación ejecutable para las distintas plataformas que soportan los smartphones y con microcontroladores ESP8266 que den funcionalidad real a las actividades gestionadas en la aplicación.

Palabras clave: aplicaciones móviles, React Native, automatización, sistemas de riego, ESP8266.

ABSTRACT

Nowadays the internet has become a necesity thing for us. It is used for the simplesst of things to the most complex. Necessities in our lives can be fulfilled on the internet.

Holding hands to internet expansion is the smartphone world, which involves everything from technology to mobile applications. Those applications are the human tools to control and take advantage of the internet.

The extensions of mobile applications are incalculable. For example, entertainment (videogames, social media…) or to control automatic systems without which the system would not function.

One of the most important areas of automation is The Agrarian system which uses mobile applications supported by the internet that can simplify and make some systems easier. It allows you to manage those systems without human involvement.

What is meant by carrying this project up, is creating a multi platform mobile application which lets the user control different watering systems and can custom their personal experience. The user will be able to schedule watering different crops at different times.

Developing this project will be using React Native tecnology which provides a funtional application in the different platforms on all smartphones. ESP8266 microcontrolators will be used to give a real time functionality.

Keywords: mobile applications, React Native, automation, watering systems, ESP8266.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE DIAGRAMAS

INTRODUCCIÓN

Desde los inicios de la humanidad, la agricultura ha sido un pilar básico de la evolución humana. A lo largo del tiempo, las distintas culturas han ido aportando avances que hasta día de hoy siguen sumándose para conseguir optimizar los procesos de cultivo.

Actualmente la automatización de los procesos mediante el uso de máquina ha ido dejando cada vez más a un lado el factor humano, haciendo para el usuario de a pie, más difícil la adaptación a las nuevas tecnologías y, vinculándose estas a empresas privadas o grandes proyectos.

Con la realización de este proyecto se pretende conseguir un acercamiento al usuario de la automatización personalizada de sus propios terrenos, llevándose a cabo los siguientes objetivos:

- Desarrollo multiplataforma de una aplicación de gestión de riego, en la cual se podrá:
    1. Añadir, eliminar o modificar campos.
    2. Añadir, eliminar o modificar cultivos.
    3. Programar riegos para los campos y cultivos seleccionados.
- Programación de microcontroladores para comprobar el correcto funcionamiento de la aplicación desarrollada.

El alcance de este proyecto abarca desde el uso de la aplicación hasta la implantación de los microcontroladores.

Las limitaciones hardware que presenta este trabajo son, que los microcontroladores utilizados son ESP8266, que tiene wifi y dos pines GPIO de entrada salida al que se conecta un relé y, que, al estar programado como servidor, al acceder a su dirección IP con una orden especificada que solo puede ser de actúa (1) o deja de actuar (0). Por lo tanto, no se puede controlar el caudal del agua directamente desde la aplicación y el ratio de actuación (sin el uso de repetidores de señal) se reduce al alcance de la señal wifi.

Para el desarrollo del proyecto se ha utilizado la tecnología de desarrollo de aplicaciones multiplataforma de React Native, la cual ha permitido desarrollar toda la parte de front-end y back-end, ayudado también de numerosas librerías y de JSON-SERVER, el cuál actúa remotamente como base de datos local.

Para la programación de los dispositivos microcontroladores se ha utilizado el entorno de Arduino y el lenguaje de Programación C++. Las pruebas se han realizado con relés de paso de corriente.

INVESTIGACIONES PREVIAS

Para el correcto desarrollo de este proyecto, se realizaron una serie de investigaciones sobre el sector en torno al cual gira el trabajo, así como de las tecnologías existentes y los componentes necesarios para llevar a cabo su correcta ejecución.

En primer lugar, se estudió de manera exhaustiva las distintas metodologías de riego que existen actualmente. Según el Dr. Ingeniero Agrónomo José Luis Fuentes Yagüe (2003) en su libro *Técnicas de Riego*, existen multitud de aspectos a tener en cuenta a la hora de llevar a cabo la ciencia de la agricultura, tales como la relación suelo-agua-planta, las necesidades hídricas de cada cultivo, la calidad del riego y otras tantas que se escapan de la competencia de este trabajo. En lo que nos compete para la recopilación de información de los cultivos y los riegos, nos centramos en la programación del riego y en los tipos de riego que existen.

Según Yagüe (2003), "la programación del riego tiene por finalidad el ahorro de agua y de energía sin reducir la producción tratando de dar una respuesta a las preguntas de:

- Cuando se debe regar.
- Qué cantidad de agua debe aplicarse a cada cultivo.
- Cuanto tiempo debe estar aplicándose el agua en cada riego.

Es por esto por lo que en proceso de creación de la aplicación se han tenido en cuenta factores determinantes para poder desarrollar un correcto programado del riego, tales como la ubicación del campo, el caudal de agua por minuto que desea regarse y la selección personalizada del tiempo que desea regarse. (Yagüe, 2003).

En todo el desarrollo de todo el proceso se da por supuesto que el usuario final que consuma los servicios de la aplicación tendrá conocimientos de los factores que necesita en sus propias tierras y en concreto en sus cultivos, dando nosotros la opción de completar algunos de estos parámetros de forma manual en la aplicación, dejando en sus manos cálculos tales como pueden ser la selección del tipo de tubería más conveniente para cada tipo de suelo y de cultivo, o el cálculo del caudal total necesario para un cultivo determinado. (Yagüe, 2003).

Caudal necesario

El caudal de agua necesaria viene dado por la expresión:

$$Q = 10 \frac{S \times Dt}{ir \times T}$$

Q = Caudal necesario, en m^3/hora.
S = Superficie regada, en ha.
Dt = Dosis total, en mm de altura de agua.
ir = Número de días empleados en regar, dentro del intervalo de riego.
T = Tiempo de riego, en horas/día.

Imagen 1. Cálculo del caudal necesario en un cultivo. Fuente: Yagüe (2003).

Los tipos de riegos más comúnmente usado, de los numerosos que podemos encontrar en revistas científicas y artículos de investigación, resaltan tres tipos principales que predominan por encima de los otros (Yagüe, 2003). Se trata de las técnicas de riego por:

- Superficie.
- Aspersión.
- Goteo.

Existen múltiples factores que diferencian a estos tipos de riego, como el tipo de válvulas utilizado, el tiempo de regado o el caudal de agua por hora. Procederemos a verlos más detalladamente a continuación.

- Riego por superficie:

Según Yagüe (2003), el riego por superficie es un sistema de riego en el cual el agua actúa siguiendo un camino marcado por tuberías o canales estratégicamente desplegados en el cual el agua fluye a través de dichas redes por actuación de la fuerza de la gravedad o por el impulso de un motor que empuja el agua a través del sistema.

Una vez el agua llega al destino puede regar por manta (el agua moja la totalidad del suelo) o por surcos, en los cuales el agua se infiltra por el costado y el fondo sin necesidad de empapar el terreno en toda su superficie.

 z

Imagen 2. Riego por surcos. Fuente: Freepick (Creative Commons).

- Riego por aspersión:

El riego por aspersión, tal y como su nombre indica es la técnica de riego donde se utilizan aspersores para producir el riego. El aspersor es un dispositivo mecánico que se encarga de impulsar el agua a gran presión y velocidad a través de él, dando como resultado final para el cultivo la sensación de una lluvia natural (creada de manera artificial por los dispositivos). El riego por aspersión presenta frente a su predecesor de superficie numerosas ventajas, tales como la mejora de la eficiencia del regado (que pasa del 40-70 % al 80%), la reducción de la mano de obra para la preparación los planos y del terreno o la eliminación de factores de obstaculización del regadío. (Yagüe, 2003).

Obviamente también posee desventajas como el elevado coste de instalación o la poco acertada precisión del riego ante condiciones meteorológicas adversas (como el viento).

Imagen 3. Riego por aspersión. Fuente: Pixabay (Creative Commons).

- Riego por goteo:

El riego por goteo (o localizado) es aquel en el que se distribuye el agua de una manera más concreta, sin un desperdicio de agua. Esto conlleva a no tener que mojarse el suelo en su totalidad, sino que simplemente utiliza pequeños caudales de agua a muy baja presión (gota a gota). El caudal en este tipo de riego suele estar por debajo de los 16L/H por metro de manguera de goteo (Yagüe, 2003).

Este sistema de riego tiene como fuerte principal el mejor aprovechamiento del agua, mayor uniformidad de riego y, por ende, mejor aprovechamiento de fertilizantes y economización de la mano de obra. Sin embargo, se necesita un personal experto para su instalación y exige un capital inicial mayor a sus dos sistemas predecesores.

Imagen 4. Riego por goteo. Fuente: Wikipedia (Creative Commons).

Dejando a un lado la recopilación de datos y parámetros que nos aporta la teoría agrícola, se realizó también un estudio de mercado para conocer de primera mano las aplicaciones existentes sobre el tema, así como su funcionamiento básico. Todas las aplicaciones están disponibles en los mercados de Android e IOS. Se irán exponiendo según el nivel creciente de proximidad a la aplicación a desarrollar en este trabajo:

- MyGarden.
- GardenAnswer.
- GlobalCampo.
- Cultivapp.
- CampoGest.
- RiegoApp.
- Siar.

Según la página oficial de MyGarden.org (2020) en *Gardening information* una aplicación en la cual puedes registrarte y tener un foro de búsqueda de información para todo tipo de plantaciones y cultivos. Solo tienes que buscar la especie en cuestión y el buscador te devolverá toda la información que posee

a cerca de la consulta (descripción general, requerimientos de cultivo, mantenimiento de la planta, características…) además de ello, puedes registrarte y compartir tus experiencias de cultivos con los demás usuarios.

En la página de GardenAnswer (2020) en *About Us* se explica de forma detallada el uso del comportamiento de la aplicación, siendo esta un identificador por imagen de toda flor, árbol o arbusto conocido. Basta con sacar una foto del cultivo deseado y la aplicación de dará una respuesta a cerca del nombre de la planta y su perfil (nombre científico y común, características de cuidado, etc).

GlobalCampo (2020) en *Inicio* indica que es una aplicación para profesionales del sector. Requiere la pertenencia al grupo Globalcaja para poder hacer un uso exhaustivo de la aplicación. En la parte visible a todos los usuarios, puede verse que es una herramienta de comparativa de precios de mercado para el cultivo, fertilizantes etc. Además de esto, proporciona un visor de satélite de las fincas y un avisador de incendios por proximidad fuego/finca. También, es una red de noticias relacionadas con el sector y enfocada a profesionales del campo.

Cultivapp es una aplicación para el registro de actividades diarias realizadas en un campo. Va dirigida a agricultores, ITAs (Ingenieros técnicos agrícolas) y cooperativas. Te permite establecer un registro escrito de las actividades que van realizándose como, el riego que se ha producido, el fertilizante que se ha esparcido, el insecticida más conveniente para el cultivo seleccionado además de incluir un chat para poder hablar con otros profesionales del sector y una lista de precios actualizada de la subasta diaria de los cultivos seleccionados, según registra la página oficial de Cultivapp (2020) en *Bienvenido a CultivAPP*. Además, incluye la posibilidad de utilizar mapas para la delimitación de las parcelas registradas en el perfil.

CampoGest es también una aplicación de gestión de cultivos para profesionales, en la cual se pueden registrar fincas, tener un cuaderno de día y ver los tratamientos y cultivos seleccionados. A diferencia de Cultivapp, requiere de la instalación previa de la empresa de ERPagro, una plataforma de estudio de rendimiento y negocio del terreno por parte de una empresa externa, según puede verse en la página oficial de Campogest (2020) en *Campogest.*

RiegoApp es una aplicación desarrollada por la empresa Irriego (2020) según se indica en *Danube &*

RiegoApp. Sin lugar a duda, es la que mayor similitud comparte con la aplicación a desarrollar en este proyecto, ya que permite el control de varios programadores desde una sola *app*, configurar riegos futuros, activar o suspender manualmente zonas de riego etc. Sin embargo, esta aplicación es de uso privado para aquellos que contraten los servicios especializados de la empresa. Esta es la encargada de hacer la instalación de los dispositivos a controlar y de darte acceso a la aplicación.

SiarApp en último lugar, es una aplicación con una funcionalidad muy similar a la que se pretende llegar con el desarrollo de este proyecto según indica El Ministerio de Agricultura, Pesca y Alimentación (2020) en *SiarApp Inicio*. Pertenece al ministerio de España y como singularidad y punto discordante con la que se va a realizar, es que dependiendo la zona (ubicación) en la que se encuentre el usuario, permite sólo determinados cultivos, en función de lo que más prospera o es más común en la zona.

Una vez se estudiaron las distintas aplicaciones del mercado más relevantes, así como su utilización se decidió seguir una serie de pautas características para el desarrollo de la aplicación multiplataforma. Tras esto, se procedió a investigar la tecnología a utilizar para el desarrollo de esta labor (React Native) y fue entonces donde se tuvo que tomar en consideración las diferentes formas de crear y desarrollar un proyecto en React Native. Existen dos formas fundamentales de crear una aplicación de código nativo. A partir de React Native-CLI o a través de Expo.

React-native-CLI (a partir de ahora, RNCLI) fue la opción que parecía más propicia en un primer momento, ya que seguía las pautas de desarrollo que habíamos trabajado a lo largo de toda la formación académica. Además de esto, las aplicaciones que se crean son mucho más ligeras y se ejecutan de manera local.

Sin embargo, con los primeros pasos usando el RNCLI, se observó que, además de requerir muchas más instalaciones de herramientas que Expo, solo podía probar la aplicación en Android ya que al trabajar con un dispositivo Windows no disponía de Xcode y en consecuencia, carecía de un simulador de plataforma IOS.

Expo es un conjunto de herramientas open source que nos ayudan a crear nuestra *app* nativa usando React Native y nos ayudan a gestionarla. En la página web puede subirse a un repositorio donde podemos hacer testing (expo.io), además de enviar la aplicación a un dispositivo físico. Este servicio de compilación que te permite crear *apps* de forma muy rápida y sencilla, además que te permite compilar tu *app* en su servidor (nube). También requiere de muchas menos instalaciones que RNCLI. Su funcionamiento es sencillo, una vez se arranca el servidor, te provee de un código QR que, a través de una aplicación previa instalada en el dispositivo, la abre en tu teléfono en tiempo real

La diferencia de Expo y RNCLI es que con Expo requieres de su aplicación para compilar tu aplicación mientras que de la otra manera la compilas tú en tu equipo. La sintaxis de creación de proyecto es muy parecida, aunque en la carpeta final se crean cosas diferentes. Por otro lado, en RNCLI se crean dos carpetas, una de Android y otra de IOS, mientras que en Expo se crea una sola carpeta de expo, necesaria para la depuración de las distintas plataformas.

En último lugar en el estudio de las consideraciones para llevar a cabo este proyecto, para darle realidad y tangibilidad se estudiaron dos mecanismos de materialización de las órdenes que se mandaban a través de la aplicación móvil. La raspberry pi 3 y el microcontrolador ESP8266.

ACLARACIÓN TRAS LA INVESTIGACIÓN

Tras realizar la tarea de recopilación de información e investigación en campos tan amplios que van desde el aprendizaje de técnicas ajenas al campo de estudio del estudiante hasta el avance en nuevas tecnologías no conocidas, se ha llegado a un conjunto de conclusiones que se tendrán en cuenta para llevar a cabo la aplicación:

1. Se utilizará como soporte de desarrollo la herramienta de Expo y no la de RNCLI, debido principalmente a la imposibilidad de realización de pruebas en entornos IOS por la falta de computadora MAC y, por ende, de la herramienta XCODE que nos brindaría un simulador.
2. La tecnología hardware a utilizar serán los microcontroladores ESP8266 y no la Raspberry Pi3. El principal motivo es la versatilidad de los microcontroladores ESP8266 y su bajo coste, de manera que, si se quisieran realizar grandes instalaciones con distintos sectores, valdría con la implantación de distintos microcontroladores conectados a sus respectivos relés con una programación idéntica, teniendo solo que cambiar en la aplicación la dirección IP destino para cada uno de los sectores. Además de esto, el coste en potencia de los relés hace que la Raspberry Pi3 soporte un número muy limitado, encareciendo mucho el proyecto si quisiera realizarse solo a escala media.
3. El tipo de riego que contemplará la aplicación será únicamente por superficie. Debido a que se requeriría el uso de un equipamiento caro y especializado, se descarta el sistema de riego por aspersión. El riego por goteo podría ser posible si se hubiera decidido utilizar como componente software la Raspberry Pi3, ya que esta cuenta con pines GPIO PWD, los cuales podrían ser capaces de controlar el caudal del agua que entra a la manguera abriendo en menor o mayor proporción las válvulas encargadas del paso del agua. Como el dispositivo a programar es el ESP8266, solo puede controlarse a través de sus pines de entrada/salida (I/O) la apertura o cierre de la válvula.
4. El tipo de aplicación será de control de riego, pudiendo administrarse diferentes fincas y cultivos, programando para cada uno de ellos los riegos que se estimen oportunos.
5. Tendrá carácter público. Cualquier persona que se descargue la aplicación y adquiera unos microcontroladores ESP8266, adecuando una programación de Arduino o adquiriendo la utilizada en este proyecto podrá ser capaz de montar su propio sistema de riego personalizado.

TECNOLOGÍA EXISTENTE

Con el fin de exponer con claridad todos los conceptos y herramientas utilizadas durante la creación de la aplicación, se procederá a desarrollarlos a continuación, desde aquellos más generales hasta los específicos y concretos.

En primer lugar, se expondrá brevemente los principios e historia de JavaScript, siendo este uno de los lenguajes de programación más relevantes para la creación de este proyecto. Como indica MDN (2020) en *JavaScript*, el lenguaje de programación JavaScript (JS) se caracteriza por ser "ligero, interpretado, o compilado justo a tiempo (just-in-time) con funciones de primera clase". Se trata de un leguaje que utiliza la secuencia de comandos (scripting) en webs, así como en otros marcos fuera de los propios navegadores, como puede ser Adobe Acrpbat, Apache CouchDB o Node.js, estando este último presente en el proyecto, por lo que se expondrá más adelante. Asimismo, algunos de las características propias de JavaScript que indica MDN (2020) son:

- El hecho de que se trata de un leguaje de que se basa en prototipos.
- Se trata de un lenguaje multiparadigma, con capacidad para ser dinámico y de un solo hilo.
- Se estableció para ser un tipo de programación enfocado en los objetivos, así como declarativa e imperativa.

Por su parte, Navarrete, T. (2006) indica en *El lenguaje JavaScript* que JavaScript es un lenguaje de programación que cuenta con características propias que lo definen:

- Utiliza programas denominados "scripts" se encuentran en las páginas HTML y son ejecutados en navegadores.
- Los scripts se basan funciones que actúan como llamadas dentro del HTML cuando algún proceso tiene lugar.

JavaScript fue creado por la organización Netscape, partiendo del lenguaje Java, con el cual mantiene

algunas similitudes, aunque también diferencias, como el hecho de que se puede ejecutar fuera del navegador. MDN (2020) añade que se trata de marcas completamente diferentes, las cuales han sido registradas por Oracle, además del hecho de que son "dos lenguajes de programación tienen sintaxis, semántica y usos muy diferentes".

Collell, J. (2013) en *CSS3 y Javascript avanzado* define Javascript como un "lenguaje de programación utilizado en el desarrollo de aplicaciones web por parte del cliente". Fue creado en el año 1995 por Netscape Corporation pero, inicialmente, su uso resultaba complicado, pues era necesario trabajar con las diferentes especificaciones de cada navegador, lo que tenía como consecuencia la obtención de un código con poca solidez. Con el fin de reducir estos problemas se crearon las bibliotecas, las cuales tenían como misión conseguir una API (*application programming interface*) que fuera común en los distintos navegadores. Según Collell, J. (2013), algunas de las ventajas que ofrecen estas bibliotecas son:

- La interactividad con el DOM.
- La interacción con los receptores utilizando el sistema de eventos.

Una de las bibliotecas más relevantes, la cual está muy relacionada con este proyecto, es React. Según la web oficial de React (2020) en *¿Qué es React?,* se trata de una biblioteca de JavaScript cuyo objetivo es la creación de interfaces de usuarios, las cuales son interactivas, además de ser de fácil creación. Asimismo, permite "componer IUs (interfaces de usuarios) complejas de pequeñas y aisladas piezas de código llamadas 'componentes'". El modo de empleo se basa en la generación de vistas sencillas para cada estado de la aplicación y, posteriormente, React realiza la actualización y renderización de los distintos componentes que son adecuados cuando se produce un cambio de datos. Asimismo, gracias a las vistas declarativas, el código obtenido puede ser depurado de forma simple.

Tal y como se indica en la web oficial de React, con esta herramienta se pueden elaborar "componentes encapsulados que manejen su propio estado", los cuales tienen la capacidad de transformarse en complejas interfaces de usuario. El hecho de pasar datos a través de la aplicación, manteniendo su estado fuera del DOM, es posible, ya que la lógica de los distintos componentes está expresada con JavaScript.

Otra de las ventajas que ofrece React, según la web de la propia herramienta, es el hecho de poder crear nuevas características sin tener que crear de nuevo el código generado. Asimismo, ofrece la posibilidad de ser renderizado desde el servidor, gracias a Node.js, así como impulsar las aplicaciones móviles con la utilización de React Native.

Esta última herramienta, React Native, ha sido un pilar fundamental a la hora de desarrollar la aplicación para este proyecto. Como se indica en la web oficial de React Native (2020), esta herramienta vincula determinadas partes del desarrollo nativo que ofrece React con la biblioteca que ofrece JavaScript de su clase, elaborando así interfaces de usuario, ya que se escribe con JavaScript pero es renderizado con código nativo. React Native permite la creación de una aplicación nativa, tanto para Android como iOS, desde cero.

Según se indica en la web oficial de React Native (2020), con React Native es posible el desarrollo de aplicaciones nativas para desarrolladores nuevos, además de garantizar que los equipos nativos presentes trabajen de manera más rápida. Este hecho viene dado por el código nativo, el cual está cubierto por los componentes que proporciona React, relacionándose además con las API nativas gracias al paradigma de UI declarativo presente tanto en React como JavaScript.

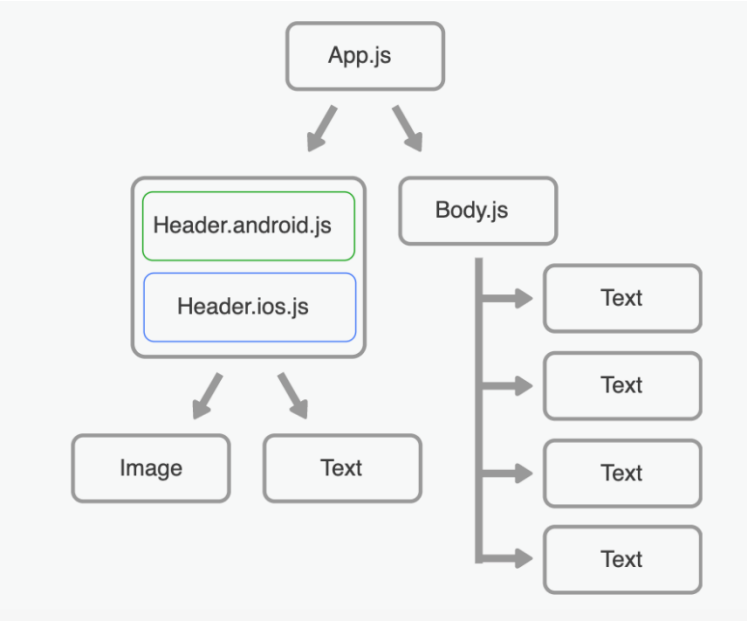

Imagen 5. Plataforma cruzada de React Native. Fuente: https://reactnative.dev/.

React Native nació en 2015, siendo lanzado por Facebook, empresa que lo ha impulsado y mejorado desde entonces, tal y como se indica en la web oficial de React Native (2020). Actualmente, React Native es mundialmente conocido, contando con la contribución de entidades y personas de todo el mundo, como Microsoft, Infinite Red, Callstack, Expo, Software mansión.

Otra parte fundamental para el desarrollo de la aplicación es Expo, una plataforma y framework

utilizado para aplicaciones universales de React, según se indica en el sitio web oficial de Expo (2020). Su principal función es la de crear, desarrollar e implementar en iOS, Android y aplicaciones web desde el propio código de JavaScript / TypeScript.

Para poder crear la aplicación multiplataforma utilizando JavaScript existen, según la página oficial de React, diferentes maneras de escribir código JavaScript, usando el convencional modelo de clase o bien usando los Hooks.

Para la ejecución de este proyecto se han utilizado los Hooks. "Los Hooks son funciones que te permiten "enganchar" el estado de React y el ciclo de vida desde componentes funcionales. Asimismo, no funcionan dentro de las clases, te permiten usar React sin clases. React proporciona algunos Hooks incorporados como useState. También puedes crear tus propios Hooks para reutilizar el comportamiento con estado entre diferentes componentes." Fragmento recuperado de la página oficial de React (https://es.reactjs.org/docs/hooks).

Para dar una explicación más visual de por qué usar hooks y no clases es simplemente porque el código es más limpio.

TouchableHighlight Class Component Example

```
class App extends Component {
  constructor(props) {
    super(props);
    this.state = { count: 0 };
  }
  onPress = () => {
    this.setState({
      count: this.state.count + 1
    });
  };
  render() {
    return (
      <View style={styles.container}>
        <TouchableHighlight onPress={this.onPress}>
          <View style={styles.button}>
            <Text>Touch Here</Text>
          </View>
        </TouchableHighlight>
        <View style={[styles.countContainer]}>
          <Text style={[styles.countText]}>
            {this.state.count ? this.state.count : null}
          </Text>
        </View>
      </View>
    );
  }
```

Imagen 6: Componente escrito como clase.

TouchableHighlight Function Component Example

```
const TouchableHighlightExample = () => {
  const [count, setCount] = useState(0);
  const onPress = () => setCount(count + 1);

  return (
    <View style={styles.container}>
      <TouchableHighlight onPress={onPress}>
        <View style={styles.button}>
          <Text>Touch Here</Text>
        </View>
      </TouchableHighlight>
      <View style={styles.countContainer}>
        <Text style={styles.countText}>
          {count ? count : null}
        </Text>
      </View>
    </View>
  );
}
```

Imagen 7: Componente escrito como hooks.

Fuente: https://reactnative.dev/docs/touchablehighlight.

Como se ha comentado anteriormente, la utilización de Node.js ha sido imprescindible. Con su utilización se pueden "crear aplicaciones network escalables", según la web oficial de Node.js (2020) en *Acerca de Node.js*. A continuación, se muestra un ejemplo de su utilización, el clásico "hola mundo", en el cual se pueden atender distintas conexiones de forma simultánea, las cuales se activan de forma independiente para devolver la llamada o callbalk.

```
const http = require('http');

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hola Mundo');
});

server.listen(port, hostname, () => {
  console.log(`El servidor se está ejecutando en http://${hostname}:${port}/`);
});
```

Imagen 8. "hola mundo" realizado con Node.js. Fuente: https://nodejs.org/es/about/.

Otra de las características de Node.js es el hecho de que los usuarios no sufren el bloqueo de proceso, pues son muy pocas las funciones de Node.js que realiza I/O de forma directa, por lo que el bloque es casi inexistente. Este hecho hace que desarrollar sistemas escalables en Node.js sea muy viable. Node.js está estrechamente relacionado con NPM, pues este proporciona el permiso para su utilización.

NPM tiene la función de administrar paquetes de Node.js. Nació en el año 2009 como proyecto de código abierto, destinado a facilitar las tareas del desarrollador de JavaScript gracias a la fácil distribución de módulos de código empaquetados, según se indica en la web oficial de NPM (2020) en About npm. El registro de NPM es una colección de acceso público a códigos fuentes abiertos disponibles para Node.js, aplicaciones de diseño web front-end, apps móviles, robots, enrutadores, así como otras muchas herramientas. NPM actúa como el cliente de línea de comandos, el cual da acceso a los desarrolladores para poder instalar y publicar los citados paquetes.

Según la web oficial de NPM (2020), los componentes que constituyen NPM son:

- El sitio web, el cual tiene el objetivo de descubrir paquetes, configurar perfiles y configurar distintos aspectos de la experiencia con npm.

- Una interfaz de línea de comando (CLI). Esta se ejecuta desde una terminal, la cual permite la interacción con el npm por parte del desarrollador.
- El registro, una base de datos pública de JavaScript, así como la metainformación en torno a él.

Son muchas las utilidades que ofrece npm, como la adaptación de paquetes de código para aplicaciones o incorporarlas tal y como son, la descarga de herramientas de forma independiente que se puede usar de forma inminente, la ejecución de paquetes sin necesidad de descarga, usando npx, compartir código con cualquier usuario de npm, sin importar su localización, etc.

Npm será un pilar para el desarrollo de este proyecto pues ayudará a instalar de forma sencilla el amplio conjunto de librerías que se han utilizado.

Otro de los pilares fundamentales es json-server que sirve para crear una API-REST y que se utilizará como base de datos local.

Para cerrar este apartado de tecnologías existentes, en cuanto a la parte hardware, se ha utilizado "el microcontrolador ESP8266, un sistema autónomo en chip integrado con pila de protocolos TCP / IP, WiFi, P2P y punto de acceso (AP)". Según indican Rosli, Habaebi e Islam (2018) en *Characteristic analysis of received signal strength indicator from esp8266 wifi transceiver module.*

Hay varios modelos de microcontrolador ESP8266, nosotros en concreto vamos a estar utilizando el ESP8266-01, que es la versión más básica y general, para la realización de este proyecto, según indican Rosli, Habaebi e Islam (2018) .

Imagen 9: Modelos de microcontroladores ESP8266. Fuente: Arjadi, R. Harry, et al.(2018).

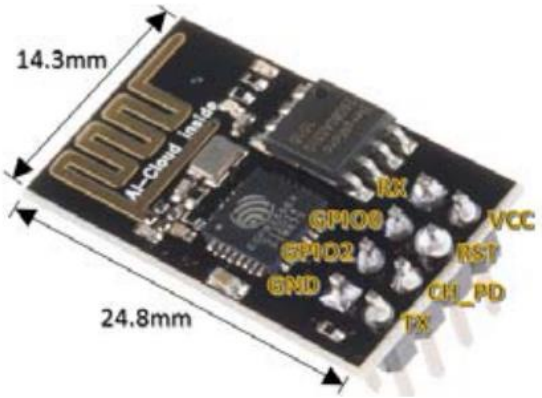

Imagen 10: Microcontrolador ESP8266-01 Fuente: Rosli, Habaebi e Islam (2018).

En la imagen 10 podemos ver que este dispositivo cuenta con 8 pines, estos son:

- GND: Toma de tierra.
- GPIO2: Entrada/Salida de propósito general.
- GPIO0: Entrada/Salida de propósito general.
- RX: Pin por donde se van a recibir los datos del puerto serie.
- TX: Pin por donde se van a transmitir los datos del puerto serie.
- CH_PD pin para apagar/encender el ESP-01: a 0 V se apaga, y a 3,3 V se enciende.
- RST pin para reiniciar (resetear) el ESP-01: si lo ponemos a 0 V (LOW) se resetea.
- VCC es por donde alimentamos el ESP-01.

Alimentaremos el ESP8266 con un adaptador USB-Serie (o TTL), por el cuál le cargaremos el programa escrito en el entorno de Arduino. Este se conectará a un enchufe inteligente a través de un relé

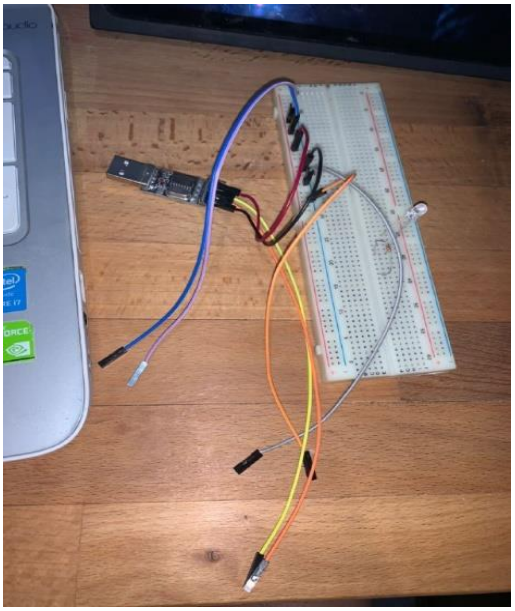

Imagen 11: USB-serie utilizado. Fuente: elaboración propia.

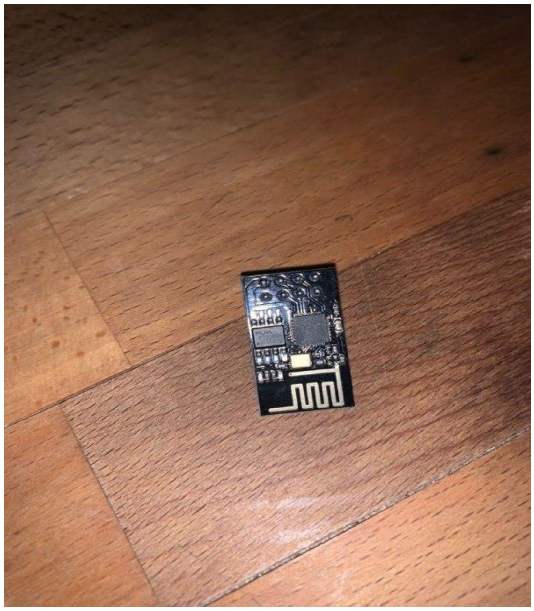

Imagen 12: ESP8266 utilizado. Fuente: elaboración propia.

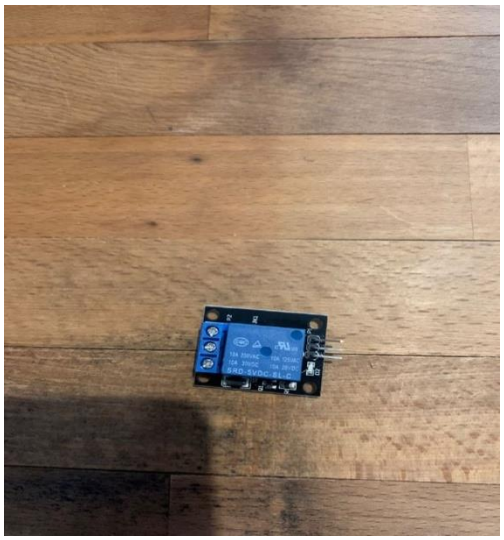

Imagen 13: Relé utilizado. Fuente: elaboración propia.

Imagen 14: Enchufe inteligente utilizado. Fuente: elaboración propia.

El programa cargado en el microcontrolador será el encargado de dar la orden de encendido o apagado, permitiendo o no el paso de corriente y, por consecuente, el encendido o apagado del motor.

PLIEGO DE CONDICIONES

En este apartado procederá a exponerse cada una de las instalaciones que hay que llevar a cabo para el correcto funcionamiento de este proyecto.

Los requerimientos para usar RNCLI en Windows son la instalación previa de Node.js, npm, Python, Chocolatey y Android Studio con sus correspondientes SDK (simuladores).

Para MAC será necesario además de npm, el uso de Xcode Command Line Tools y Watchman, Cocoapods y Homebrew (puede instalarse también Android Studio para probar en ambos sistemas operativos desde simuladores virtuales).

Para el uso de Expo (que según las aclaraciones es lo que se usará para este trabajo) necesitaremos simplemente instalar Node.js, npm y la aplicación de expo en nuestro terminal físico en el que queramos probar la aplicación.

El desarrollo de la aplicación se ha llevado a cabo utilizando el método de trabajo "de administrador" de Expo, esto es, utilizando expo-cli, la aplicación de Expo (cliente) y el manejo de APIs a las que hacerle peticiones de get, post, put y delete.

No se usan entornos de desarrollo como Xcode o AndroidStudio, solo escribe código JavaScript y CSS para darle estilos a la aplicación programada.

Una de las principales diferencias de usar expo-cli a react-native cli, es que, en la creación del proyecto, en el primero se nos crea una carpeta Expo y en segundo dos carpetas diferenciadas de IOS y Android, permitiendo hacer las instalaciones de las librerías necesarias de manera simultáneas. No obstante, expo nos proporciona un comando (run expo eject) que una vez terminado el proyecto, lo descompone para darnos el control total sobre el SDK(kit de desarrollo software) y la aplicación.

El proceso para la instalación de Expo es simple:

1. Descarga de Node.JS. Este elemento es fundamental para poder hacer uso del ecosistema JavaScript, incluyendo Expo y React Native. Para poder sacar el máximo potencial a la herramienta es recomendable descargar la última versión de Node.js.

2. Posteriormente, será necesaria la obtención las herramientas de línea de comandos npm. Para poder guardar en paquetes y publicar proyectos localmente es fundamental esta herramienta. Se descarga por defecto al descargar Node.js.
3. A partir de este momento ya es posible la creación del primer proyecto, siendo necesaria la obtención de la aplicación de Expo en un dispositivo físico.
4. Para poder previsualizar el proyecto es necesario abrir Expo Client en el dispositivo y escanear el código QR generado, ya sea con iOS o Android.
5. Finalmente, se puede empezar a generar el código con el editor, ya sea Atom, VSCode, Sublime Text, etc.

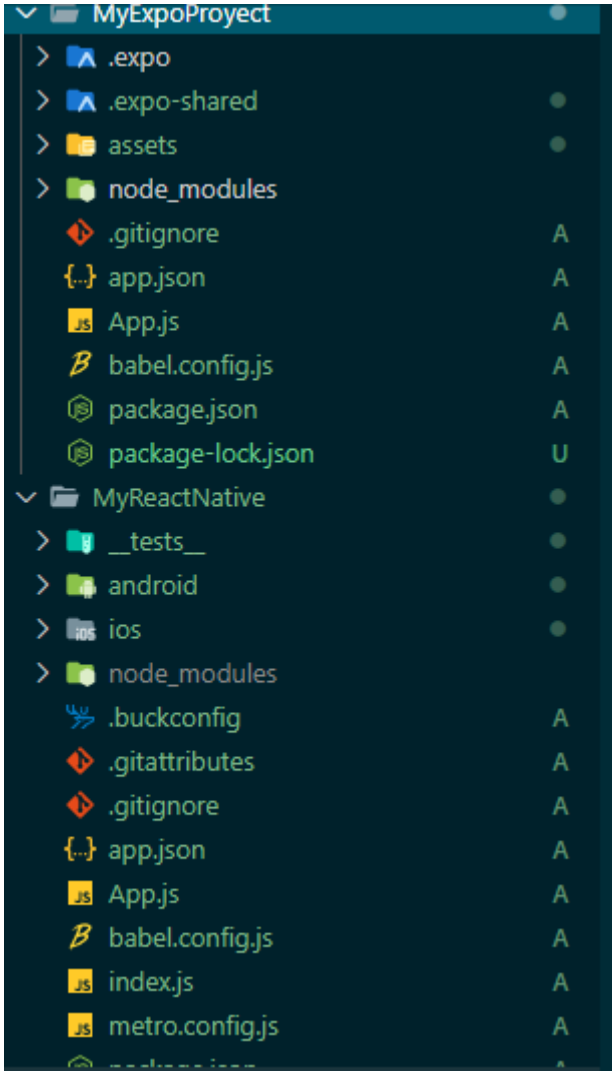

Imagen 15: Diferencias de carpetas de proyectos creados con expo-cli o con react-native-cli. Fuente: elaboración propia.

Como único prerrequisito exigido por Expo, será tener Node.js instalado en el ordenador, para ello, simplemente tenemos que visitar su página oficial (https://nodejs.org/es/) y ejecutar una descarga (la

versión LTS es la recomendada ya que es aquella que asegura una estabilidad total, mientras que la actual, aun siendo más nueva, está en continuo desarrollo y puede contener incompatibilidades) .Con la instalación de node, viene asociado npm. Una vez hecho esto, podemos comprobar que la instalación se ha llevado a cabo correctamente y comprobar la versión abriendo el cmd y escribiendo el comando "node -v".

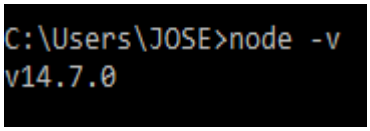

Imagen 16: Comprobación de instalación de node. Fuente: elaboración propia.

Una vez confirmado que tenemos node instalado, procederemos a instalar la herramienta de desarrollo local de expo (CLI Command line interface). Basta para ello con ejecutar la orden:

" npm install -g expo-cli ".

Podemos comprobar que la instalación se ha llevado a cabo con éxito ejecutando la orden "expo -V".

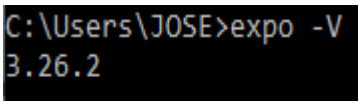

Imagen 17: Comprobación de instalación de expo. Fuente: elaboración propia.

A continuación, instalaremos el cliente en nuestro dispositivo físico. Según la página oficial de expo (https://docs.expo.io/), estos clientes pueden encontrarse fácilmente en los mercados a través de los siguientes enlaces:

IOS:
https://search.itunes.apple.com/WebObjects/MZContentLink.woa/wa/link?path=apps%2fexponent

Android:

https://play.google.com/store/apps/details?id=host.exp.exponent

Con todo el entorno preparado, podemos proceder a la creación de nuestro proyecto. Solamente tenemos que abrir la terminal del ordenador y teclear: "expo init Nombreproyecto ". Esta sentencia creará una carpeta en el directorio en el que estemos ubicado con el nombre del proyecto que deseemos.

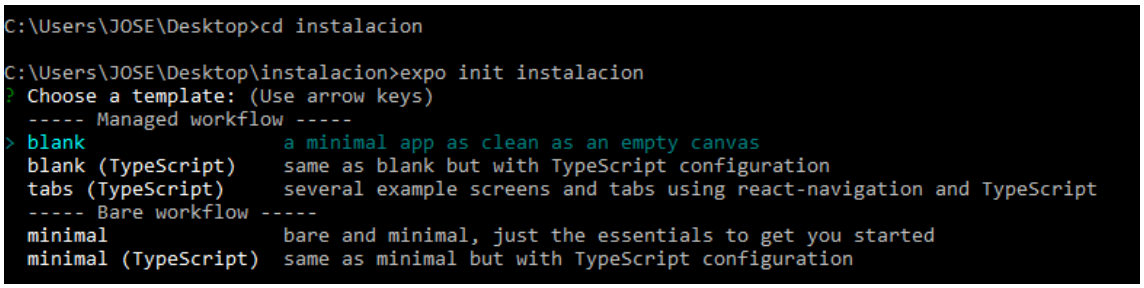

Imagen 18: Opciones de las plantillas del proyecto. Fuente: elaboración propia.

Para que se lleve a cabo esta creación, debemos previamente seleccionar una de las plantillas que nos ofrece en línea de comando. Para este proyecto se ha empezado con la seleccionada en color azul, es decir, en blanco.

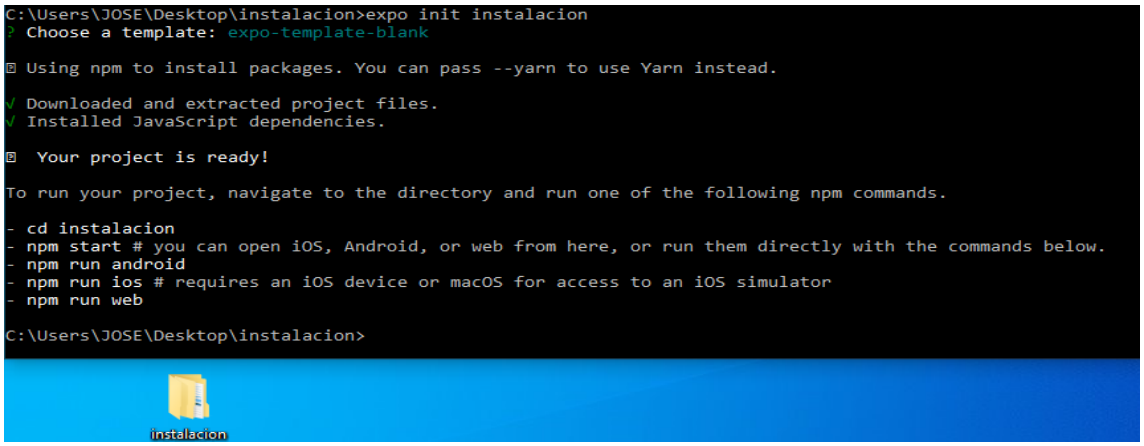

Imagen 19: Correcta instalación de las dependencias del proyecto. Elaboración propia.

Una vez ha terminado la instalación, nos da las instrucciones de cambiarnos al directorio destino y ejecutar las ordenes de npm start para ejecutar desde IOS, Android o web o alguna de las plataformas en concreto. Estas requerirán de un simulador arrancado para poder funcionar. Como para el desarrollo de este proyecto trabajaremos con Expo, tendremos que ejecutar la orden de npm start dentro del directorio.

Según la página oficial de expo (https://docs.expo.io/), "Expo CLI iniciará Metro Bundler, que es un servidor HTTP que compila el código JavaScript de nuestra aplicación usando Babel y lo sirve a la aplicación Expo. También muestra Expo Dev Tools, una interfaz gráfica para Expo CLI".

Imagen 20: Resultado de la orden npm start. Fuente: elaboración propia.

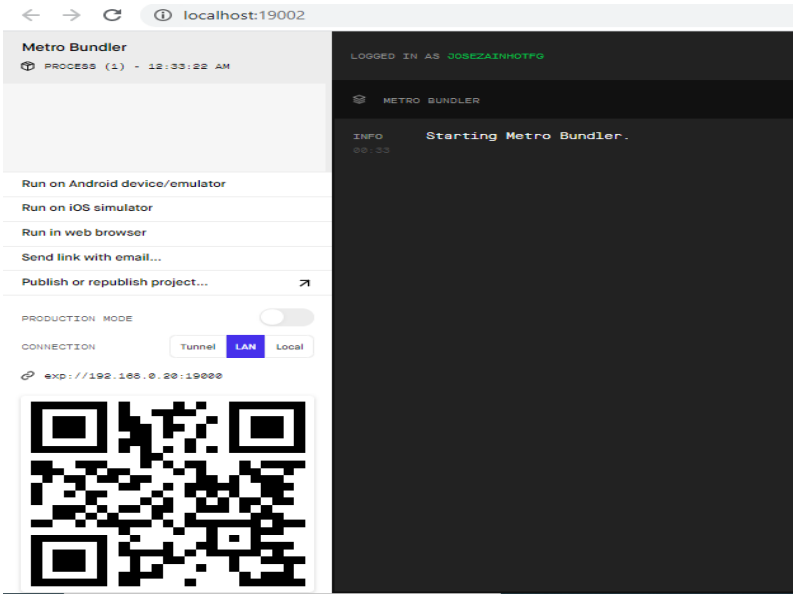

Imagen 21: Expo Dev Tools. Fuente: elaboración propia.

Una vez escaneamos el código QR proporcionado en el cmd o en el expo debe tools con la cámara del Iphone o desde dentro de la aplicación de Expo de Android, se nos abrirá en el terminal una pantalla de bienvenida en nuestro terminal. La pantalla principal App.js. desde esa pantalla se partirá para la programación total de la aplicación.

Para el resto de las dependencias, se utilizará npm seguido de la dependencia a instalar. Como ejemplo esclarecedor y seleccionado por su vital importancia, para instalar el json-server, se ejecutará por línea de comando: “ npm i json-server”.

COMPONENTES UTILIZADOS

- DE REACT:
    1. React.
    2. UseState.
    3. UseEffect.
- DE REACT NATIVE:
    1. StyleSheet.
    2. View.
    3. Alert,
    4. Platform.
    5. Keyboard,
    6. TouchableWithoutFeedback.
    7. TouchableHighlight.
    8. Text.
    9. FlatList.
    10. Button.
    11. Image.
    12. Dimensions.
    13. ScrollView.
    14. TextInput.
- DE NATIVE-BASE:
    1. Container.
    2. Button.
    3. Text.
    4. H1.
    5. Form.
    6. Item.
    7. Input.
    8. Root.
- DE REACT-NATIVE-PAPER:
    1. Checkbox.
    2. List.

- DE AXIOS:
    1. Axios.
- DE REACT-NAVIGATION-NATIVE:
    2. UseNavigation.
    3. NavigationContainer.

- DE REACT-NATIVE-MODAL-DATE-TIME-PICKER:
    1. DateTimePickerModal.
- DE REACT-NAVIGATION-STACK:
    1. CreateStackNavigator.
- DE REACT-NAVIGATION-CHAR-KIT:
    1. LineChar.

DEPENDENCIAS REQUERIDAS

"@react-native-community/datetimepicker": "^2.4.0".

"@react-native-community/masked-view": "0.1.10".

"@react-native-community/picker": "1.6.0".

"@react-navigation/native": "^5.7.3".

"@react-navigation/stack": "^5.9.0".

"axios": "^0.19.2".

"dotenv": "^8.2.0".

"expo": "~38.0.8".

"expo-font": "~8.2.1".

"expo-status-bar": "^1.0.2".

"moment": "^2.27.0".

"native-base": "^2.13.13".

"react": "~16.11.0".

"react-chartjs-2": "^2.10.0".

"react-dom": "~16.11.0".

"react-native": https://github.com/expo/react-native/archive/sdk-38.0.2.tar.gz.

"react-native-gesture-handler": "~1.6.0".

"react-native-modal-datetime-picker": "^8.9.3".

"react-native-multiple-select": "^0.5.5".

"react-native-paper": "^4.0.1".

"react-native-reanimated": "~1.9.0".

"react-native-safe-area-context": "~3.0.7".

"react-native-screens": "~2.9.0".

"react-native-web": "~0.11.7".

"react-native-chart-kit": "^6.6.0".

DESARROLLO DE LA APLICACIÓN

INTRODUCCIÓN

Una vez finalizada la aplicación, las ordenes que se envían a través de las acciones llevadas a cabo, serán publicadas en un el servidor de json-server con las peticiones a axios como se verá más adelante. En este servidor se almacenará toda la información necesaria para el correcto funcionamiento y, además es desde ese servidor desde donde serán leídos los datos necesarios para la programación de los riegos por parte del microcontrolador, permitiendo así el paso de corriente (encendido/apagado) a las horas establecidas.

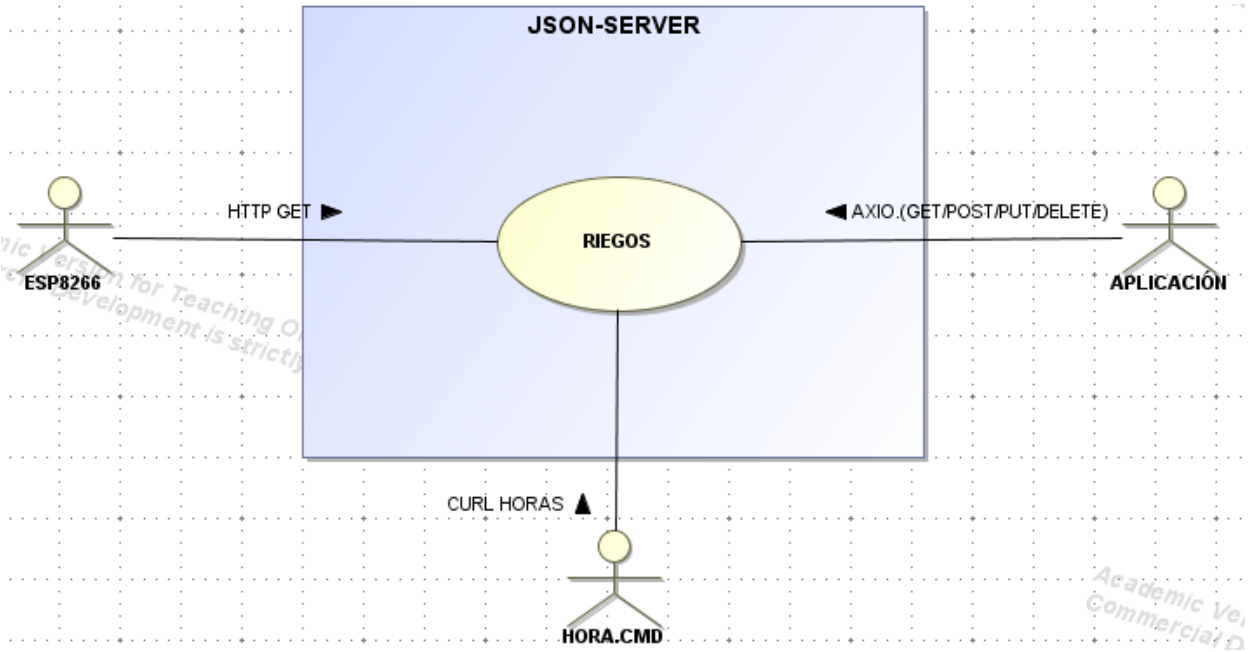

Diagrama1: Descripción general del sistema. Fuente: Elaboración propia.

Por otra parte, hay que aclarar que la aplicación tendrá estructura de árbol, es decir, se podrá ir añadiendo cosas a medida que se profundice en la estructura. En la pantalla de inicio se nos listan los campos. Dentro de 1 campo puede haber n cultivos y, a su vez, dentro de este puede haber n riegos.

La primera vez que se inicia la aplicación no habrá nada en el servidor, por lo que la única acción que podemos hacer es la de crear un campo. No se pueden crear cultivos ni riegos si no existe un campo.

Una vez creado el primer campo, podremos crear diferentes cultivos y, de igual manera que antes, cada uno de esos cultivos podrá tener el número de riegos que desee.

Existen también dos pantallas especiales dentro de campo y de cultivo. Estas son meteorología (dentro de campo) que nos permite obtener la previsión para un campo (con una ubicación establecida y que, por ende, comparten todos los cultivos y riegos) y estadísticas (dentro de cada cultivo se podrá tener un conocimiento del caudal, agua y minutos programados de todos los riegos que engloba ese mismo cultivo).

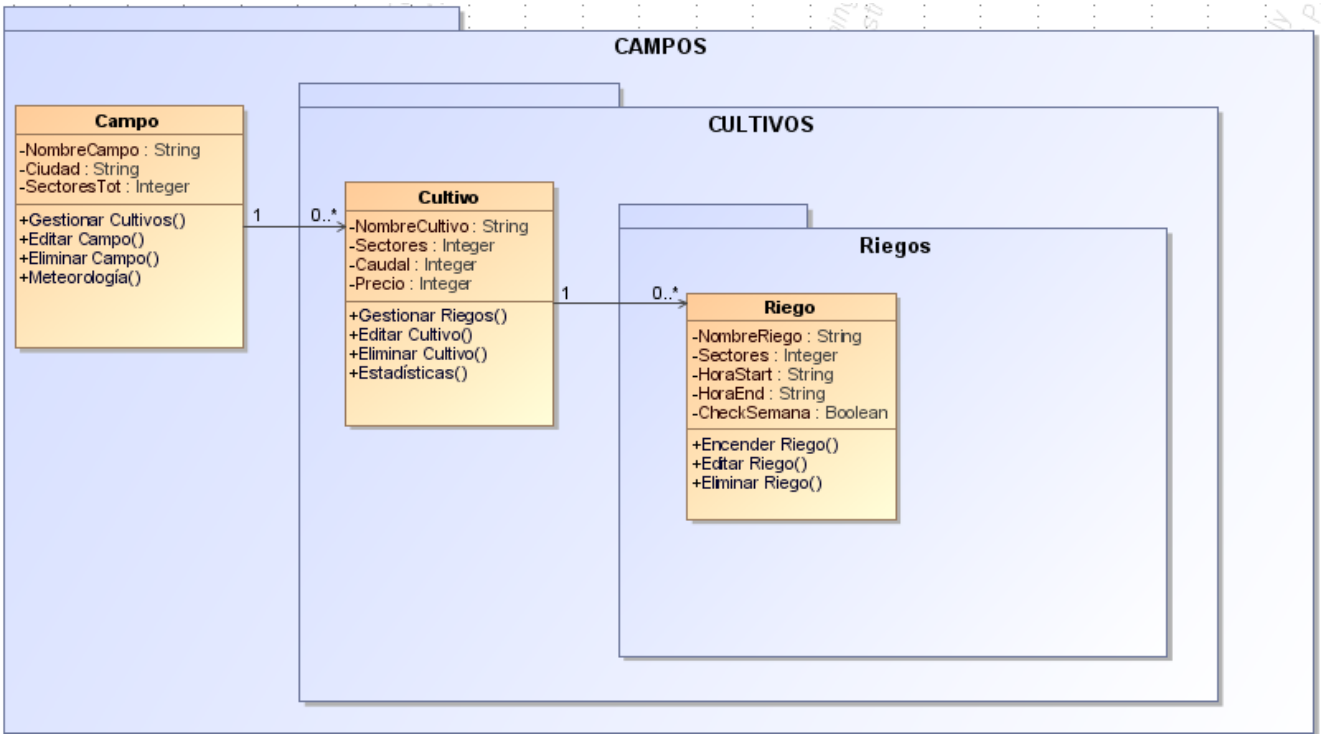

Diagrama 2: Estructura de los archivos. Fuente: Elaboración propia.

APP.JS

App.js es el fichero principal para la ejecución de la aplicación. Es el nodo raíz (root) todo el desarrollo. Es fundamental también el uso de librerías y, especialmente de React Navigation. React Navigation nos permite navegar entre pantallas o ficheros a través del envío de peticiones. El modo utilizado en este trabajo es mediante la función onPress() de los botones establecidos. Las numerosas librerías se irán explicando a medida que aparezcan en el código y es por ello que empezamos con React Navigation. Esta es una librería especial que tiene que ir declarada en la primera línea del fichero App.js.

```
“import "react-native-gesture-handler";
```

Su instalación en línea de comandos se hará con la orden: “npm install @react-navigation/native”

Nos permite declarar las diferentes pantallas que tendrá nuestra aplicación. Para ello, debe importarse la clase y hacer la declaración de la misma dentro de un StackScreen dentro este de un NavigationContainer. Estos son componentes propios de la librería de React Navigation por lo que habrá que importarlos también.

```
import { NavigationContainer } from "@react-navigation/native";
import { createStackNavigator } from "@react-navigation/stack";
const Stack = createStackNavigator();
import Login from "./views/Login";
```

.

.

.

```
const App = () => {
  return (
    <>
      <Root>
        <NavigationContainer>
          <Stack.Navigator initialRouteName="Login">
            <Stack.Screen
              name="Login"
              component={Login}
              options={{
                title: "Iniciar Sesión",
                headerShown: false,
              }}/>
```

Tal y como puede verse en las propiedades declaradas anteriormente en el Stack.Navigator, el nombre de la pantalla sobre la que se hará la primera carga será la pantalla de “Login”.

Login.js

Imagen 22: Login.js. Fuente: Elaboración propia.

En la pantalla de Login, tenemos en la pantalla definido un H1 definido en la librería de native-base con el texto del nombre de la aplicación y un botón con el texto de "Comenzar" con una propiedad

```
onPress={() => navigation.navigate("InicioCampos")}
```

que nos llevará a la siguiente pantalla cuando se pulse el botón. En este caso concreto, la siguiente pantalla es InicioCampos.

InicioCampos.js

Imagen 23: InicioCampos.js. Fuente: elaboración propia.

En esta pantalla tenemos definido un componente de texto indicando que se está en el "Menú campos", un botón de Añadir campo que nos conduce a la pantalla de NuevoCampo y un componente FlatList que irá listando cada uno de los campos que vayamos añadiendo a la base de datos, para lo cual se utilizará la librería de axios.

Axios es una librería Javascript que facilita todo tipo de operaciones como cliente HTTP. Con Axios se pueden realizar peticiones de tipo get, post, put y delete a un servidor.

En este proyecto se está utilizando json-server que sirve para crear una API-REST y que estamos utilizando para el almacenamiento de la información relativa a los campos, cultivos y riegos.

Para mostrar los elementos del FlatList se está haciendo una solicitud de tipo get al servidor definido en el proyecto como db.json.

```
 const resultado = await axios.get(`http://${ipserver}:3000/campos`);
          guardarCampos(resultado.data);
```

Cada ítem del FlatList tiene definido un OnPress que nos conducirá a una pantalla de DetallesCampos específica de cada ítem

```
onPress={() =>
                navigation.navigate("DetallesCampo", {
                  item,
                  guardarConsultarAPI,
                })
              }
```

También tiene programado una función ternaria que, en caso de que aún no se haya añadido ningún campo, nos muestre un texto diferente en la cabecera del FlatList, invitando a crear un nuevo campo

```
<Text style={styles.headline}>
          {campos.length > 0 ? "Mis campos" : "Aun no hay Campos"}
        </Text>
```

NuevoCampo.js

Imagen 24: NuevoCampo.js. Fuente: elaboración propia.

Esta pantalla está compuesta principalmente por tres componentes de tipo TextImput y un botón de guardado. Un campo está formado por la composición de estos valores. El valor de las variables de cada uno de los TextInput se inicializa como vacío y, cuando el usuario los rellena y pulsa el botón, este llama a la función guardarCampo definida en su propiedad OnPress() y se almacenan dichos valores en el servidor mediante una petición axios post en el json-server. Además, en esta función está definida una redirección a la pantalla de InicioCampos que mostraría el campo añadido en el FlatList.

```
const campo = { nombre, ciudad, sectoresTot };
```

```
const guardarCampo = async () => {
await axios.post(`http://${ipserver}:3000/campos`, campo);
 <Button onPress={() => guardarCampo()}
```

DetallesCampo.js

Imagen 25: DetallesCampo.js. Fuente: elaboración propia.

En esta pantalla, tal y como su nombre indica, se muestran los detalles del campo sobre el que hemos pulsado en el FlatList. Se muestra el nombre como título principal y la ubicación y el número de sectores como Texto. Pueden observarse cuatro botones, cada uno de ellos con una funcionalidad.

1. Eliminar campo: al pulsar sobre el botón de eliminar, mediante el onPress() definido en el botón se llama a una función de mostrarConfinmacion, en la cual mediante un componente Alert nos notifica por pantalla que estamos a punto de eliminar un campo, una descripción con las consecuencias y dos opciones, de confirmar o de cancelar. Si pulsamos la de cancelar, se cierra la notificación sin realizar nada, pero si se pulsa en confirmar, se llama a la función elimnarCampo() que mediante una petición axios.delete elimina el campo del servidor y por consecuente deja de mostrarse en el FlatList de InicioCampo, pantalla a la que se nos redirige una vez pulsado la confirmación de la eliminación.

```
const mostrarConfimacion = () => {
    Alert.alert("¿Desea eliminar el campo?",
      "Asegurese de haber eliminado los cultivos y riegos",
      [
        { text: "Si, eliminar", onPress: () => eliminarCampo() },
        { text: "Cancelar", style: "cancel" },
      ]);};
```

Imagen 26: DetallesCampo.js. Alerta al pulsar eliminar. Fuente: elaboración propia.

2. Editar campo: al pulsar sobre este botón, se pasa a la pantalla de NuevoCampo, se realiza como antes mediante la propiedad onPress() del botón. Se pasa como parámetros el estado de la consulta a la API y los valores que componen el campo para que aparezcan como escritos y puedan editarse. Al pulsar de nuevo sobre el botón de guardado, se actualizan los valores en el FlatList.

```
onPress={() =>navigation.navigate("NuevoCampo", {
campo: route.params.item, guardarConsultarAPI, })}
```

Para determinar si se está editando un campo o añadiendo uno nuevo, existe una función definida con el componente useEffect que determina si los valores del campo están vacíos (modo de adición de nuevo campo) o si el estado de los valores está lleno (modo de edición). Si se encuentra en estado de edición, la solicitud axios que se realiza es de tipo put, que lo que hace es modificar un valor ya existente.

```
useEffect(() => {
    if (route.params.campo) {
      const { nombre, ciudad, sectoresTot } = route.params.campo;
      guardarNombre(nombre);
      guardarCiudad(ciudad);
      guardarSectoresTot(sectoresTot);
    } }, []);
```

```
if (route.params.campo) {
      const { id } = route.params.campo;
      campo.id = id;
      const url = `http://${ipserver}:3000/campos/${id}`;
        try {
          await axios.put(url, campo);
```

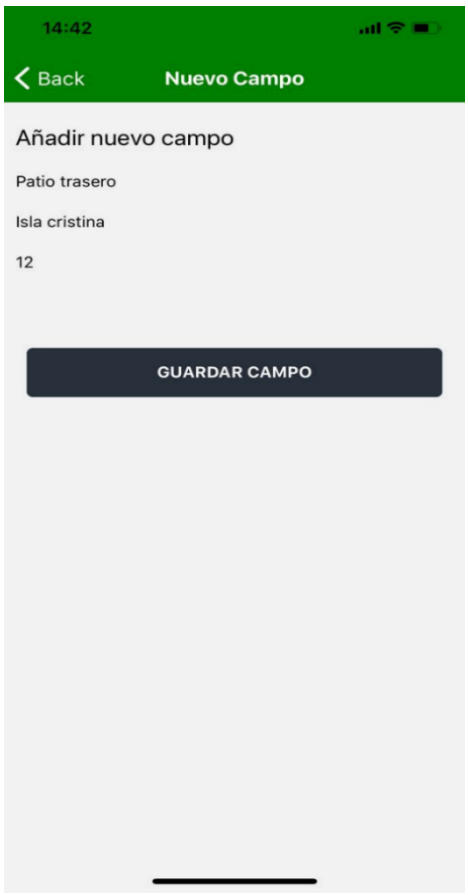

Imagen 27: NuevoCampo.js al pulsar editar. Fuente: elaboración propia.

3. Meteorología: Al pulsar sobre este botón, el sistema nos conduce a una pantalla en la que se encuentran definidos dos botones que están llamando a esta función.

Al pulsar sobre ellos, se hace una llamada a una API que nos proporciona el pronóstico diario o la previsión semanal, guardando el resultado y permitiendo extraer la información. La diferencia entre el primero y el segundo botón es simplemente la llamada a la API.

Pulsando el primer botón:

```
useEffect(() => {
    const consultarClima = async () => { if (consultar) {
        console.log(ciudad);
        const appId = "7622cfadb0870e41f0ff4d35d225cf17";
        const url = `http://api.openweathermap.org/data/2.5/weather?q=${c
iudad},{ES}&appid=${appId}`;
      try {
          const respuesta = await fetch(url);
          const resultado = await respuesta.json();
          guardarResultado(resultado);
          guardarLongi(resultado.coord.lon);
          guardarConsultar(false);
        } catch (error) {
          mostrarAlerta();}}};
      consultarClima(); }, [consultar]);
```

Pulsando el segundo botón:

```
useEffect(() => {
    const consultarClima7 = async () => { if (consultar7) {
        const appId = "7622cfadb0870e41f0ff4d35d225cf17";
        const url2 = `http://api.openweathermap.org/data/2.5/weather?q=${
ciudad},{ES}&appid=${appId}`;
        const respuesta2 = await fetch(url2);
        const resultado2 = await respuesta2.json();
        const url7 = `https://api.openweathermap.org/data/2.5/onecall?lat
=${resultado2.coord.lat}&lon=${resultado2.coord.lon}&exclude=hourly,minut
ely&appid=${appId}`;
        try {
          const respuesta7 = await fetch(url7);
          const resultado7 = await respuesta7.json();
          guardarResultado7(resultado7);
          guardarConsultar7(false);
        } catch (error) {
          mostrarAlerta7(); }}};
    consultarClima7();
  }, [consultar7]);
```

Puede verse que en esta segunda llamada se hace también una llamada a la primera. Esto es debido a que se requiere la latitud y longitud de la ciudad escrita y viene dado como parámetro.

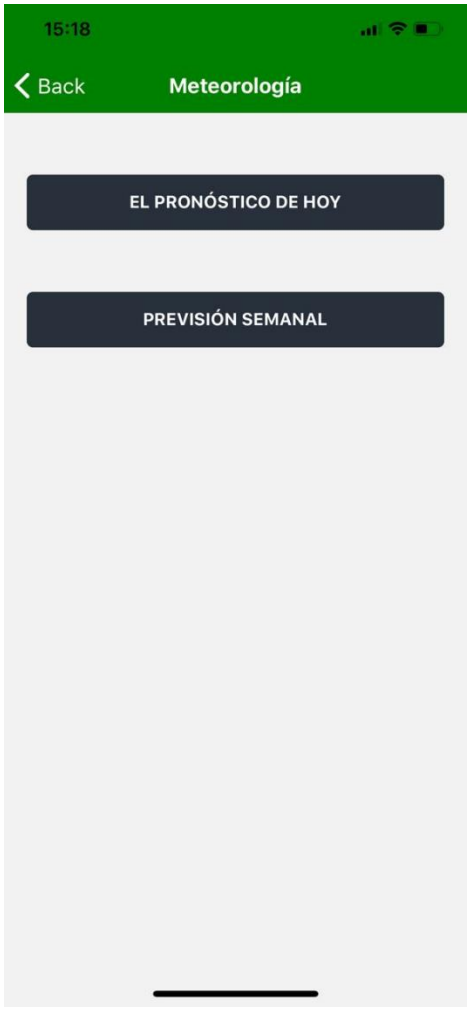

Imagen 28: GrafiCampo.js

Lo que podemos observar en esta imagen (figura X+1) es la recopilación de la respuesta de la API. Recogemos la temperatura actual, la máxima, la mínima, el nombre de la ciudad (tal y como viene devuelto de la API) así como los valores de humedad. Se está recogiendo también el icono y se está mostrando como imagen.

```
<Text style={[styles.texto, styles.actual]}>
        {parseInt(main.temp - kelvin)}
        <Text style={styles.temperatura}>&#x2103;</Text>
        <Image
          style={{ width: 66, height: 58 }}
          source={{
            uri: `http://openweathermap.org/img/w/${resultado.weather[0].
icon}.png`,
          }}
        />
      </Text>
```

Imagen 29: GrafiCampo.js botón "El pronóstico de hoy".

En la siguiente imagen podemos observar como se muestra en formato de imagen todos los iconos proporcionados por la API de la precipitación en formato visual. Posteriormente se ha elaborado una gráfica con el componente LineChar de la librería react-native-char-kit, pasando los valores medios, máximos y mínimos de la temperatura. También se ha usado la librería momentjs para determinar el día de la semana en el que se está y poder imprimir el eje X de forma correcta. Por último, tambiés se muestra una gráfica del factor de humedad.

```
<LineChart
          bezier
          yAxisLabel={"ºC"}
          data={{
            labels: semana,
            datasets: [
              {
                data: [uno, dos, tres, cuatro, cinco, seis, siete],
                strokeWidth: 2,
                color: (opacity = 1) => `rgb(255, 255, 255, ${opacity },
              {
                data: [unox, dosx, tresx, cuatrox, cincox, seisx, sietex],
```

```
            strokeWidth: 2,
            color: (opacity = 1) => `rgba(255,0,0,${opacity})`, },
          {
           data: [unom, dosm, tresm, cuatrom, cincom, seism, sietem],
            strokeWidth: 2,
            color: (opacity = 1) => `rgba(0,0,0, ${opacity})`,},],
        legend: ["Media", "Máxima", "Mínima"], }}
      width={Dimensions.get("window").width}
      height={220}
      chartConfig={{
        backgroundColor: "green",
        backgroundGradientFrom: "green",
        backgroundGradientTo: "green",
        decimalPlaces: 0, // optional, defaults to 2dp
        color: (opacity = 0) => `rgba(255, 255, 255, ${opacity})`,
        style: {
          borderRadius: 300, }, }}
      style={{
        marginVertical: 8,
        marginRight: 18,
        borderRadius: 18,
      }}  />
```

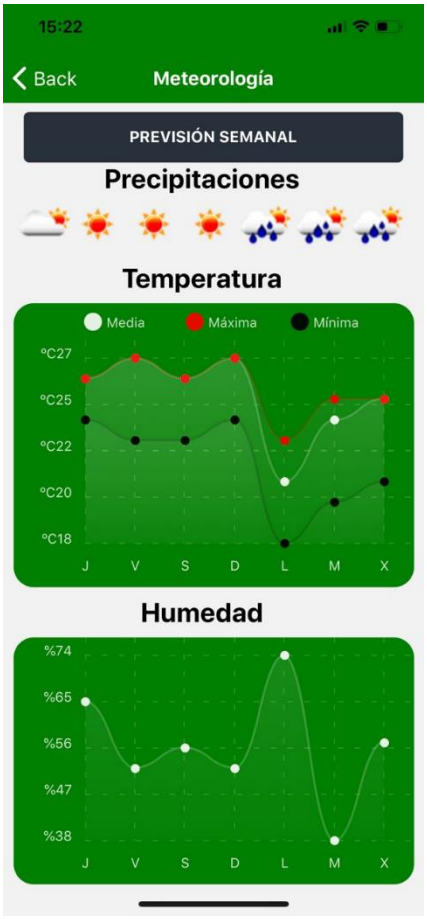

Imagen 30 GrafiCampo.js. botón "Previsión semanal".

4. Gestionar Cultivos: al pulsar sobre este botón, se pasa a la siguiente pantalla de InicioCultivo, se realiza mediante la propiedad onPress() del botón. Además, se pasa como parámetros el estado de la consulta a la API y los valores que componen el campo

```
onPress={() => navigation.navigate("InicioCultivo", {
campo: route.params.item, guardarConsultarAPI,})}
```

InicioCultivo.js

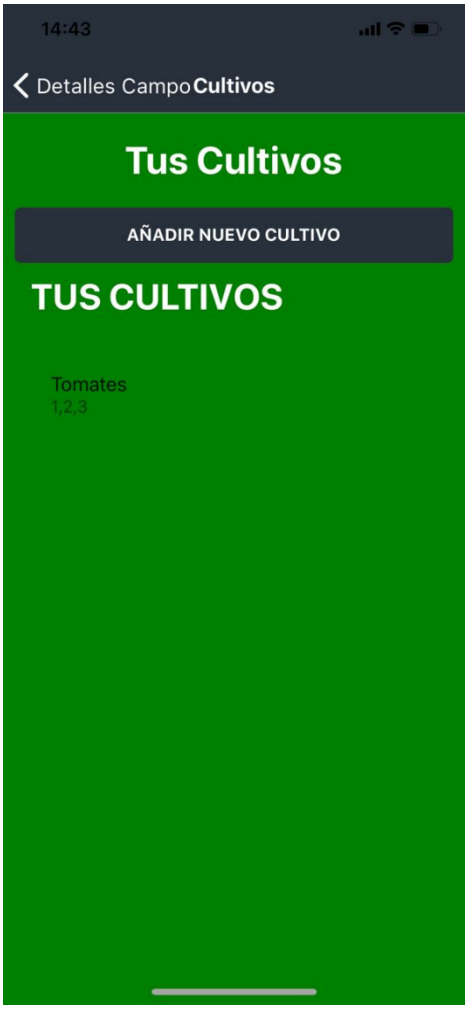

Imagen 31: InicioCultivo.js. Fuente: elaboración propia.

En esta pantalla tenemos definido un componente de texto indicando que se está en una subsección del campo "Tus Cultivos", un botón de Añadir nuevo cultivo que nos conduce a la pantalla de NuevoCultivo y un componente FlatList que irá listando cada uno de los cultivos que vayamos añadiendo al servidor.

Para mostrar los elementos del FlatList se está haciendo de nuevo una solicitud de tipo get al servidor db.json, esta vez con la particularidad de que se le está pasando también como parámetroel Id del campo del que estamos dentro, haciendo así posible que cada campo tenga unos cultivos determinados. De no hacerlo así en cada campo se mostrarían todos los cultivos existentes.

```
const obtenerCultivosApi = async () => {
      try {
          const resultado = await axios.get(
```

```
            `http://${ipserver}:3000/cultivos/?idcampo=${idcampo}`;
             guardarCultivos(resultado.data);
```

Cada ítem del FlatList tiene definido un OnPress que nos conducirá a una pantalla de DetallesCultivo específica de cada ítem y se cuenta también tiene programado una función ternaria que, en caso de que aún no se haya añadido ningún cultivo, nos muestre un texto diferente en la cabecera del FlatList.

```
<Text style={styles.headline}>
            {cultivos.length > 0 ? "Tus cultivos" : "Aun no hay Cultivos"
}
          </Text>
```

NuevoCultivo.js

Imagen 32: NuevoCultivo.js. Fuente: elaboración propia.

Esta pantalla está compuesta de nuevo por tres componentes de tipo TextImput y un botón de guardado. Un cultivo está formado por la composición de estos valores. El valor de las variables de cada uno de los TextInput se inicializa como vacío y, cuando el usuario los rellena y pulsa el botón, este llama a la función guardarCultivo definida en su propiedad OnPress() y se almacenan dichos valores en el servidor mediante una petición axios post en el json-server. Además, en esta función está definida una redirección a la pantalla de InicioCultivo que mostraría el cultivo añadido en el FlatList.

```
await axios.post(`http://${ipserver}:3000/cultivos`, cultivo);
```

DetallesCultivo.js

Imagen 33: DetallesCultivo.js. Fuente: elaboración propia.

En esta pantalla, se muestran los detalles del cultivo sobre el que hemos pulsado en el FlatList. Se muestra el nombre como título principal los sectores y el caudal/hora como Texto. Pueden observarse cuatro botones, cada uno de ellos con una funcionalidad.

1. Eliminar cultivo: al pulsar sobre el botón de eliminar, mediante el onPress() definido en el botón se llama a una función de mostrarConfinmación, en la cual mediante un componente Alert tal y como pasaba anteriormente en la pantalla de DetallesCampos. Si se confirma la eliminación se llama a la función elimnarCultivo() que, mediante una petición axios.delete elimina el cultivo y deja de mostrarse en el FlatList de InicioCultivo,

```
const mostrarConfimacion = () => {
    Alert.alert(
      "¿Desea eliminar el cultivo?",
```

```
      "Asegúrese de no tener riegos programados",
      [
        { text: "Si, eliminar", onPress: () => eliminarCultivo() },
        { text: "Cancelar", style: "cancel" },
      ] ); };
```

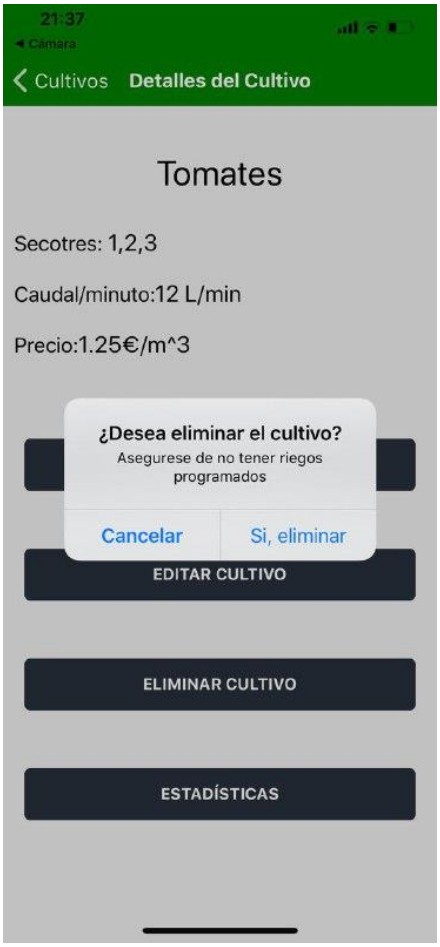

Imagen 34: DetallesCultivo.js. Alerta al pulsar eliminar. Fuente: elaboración propia.

2. Editar cultivo: al pulsar sobre este botón, se pasa a la pantalla de NuevoCultivo, se realiza como antes mediante la propiedad onPress() del botón. Se pasa como parámetros el estado de la consulta a la API y los valores que componen el cultivo para que aparezcan como escritos y puedan editarse. Al pulsar de nuevo sobre el botón de guardado, se actualizan los valores en el FlatList.

```
onPress={() =>
          navigation.navigate("NuevoCultivo", {
            cultivo: route.params.item, idcampo, guardarConsultarAPI,}) }
```

De nuevo para determinar si se está editando un cultivo o añadiendo uno nuevo, existe una función que determina si los valores del cultivo están vacíos (creando el cultivo) o si el estado de los valores está lleno (modo de edición). Si se encuentra en estado de edición, la solicitud axios que se realiza es de tipo put.

```
if (route.params.cultivo) {
      const { id } = route.params.cultivo;
      cultivo.id = id;
      const url = `http://${ipserver}:3000/cultivos/${id}`;
        try {
          await axios.put(url, cultivo);
```

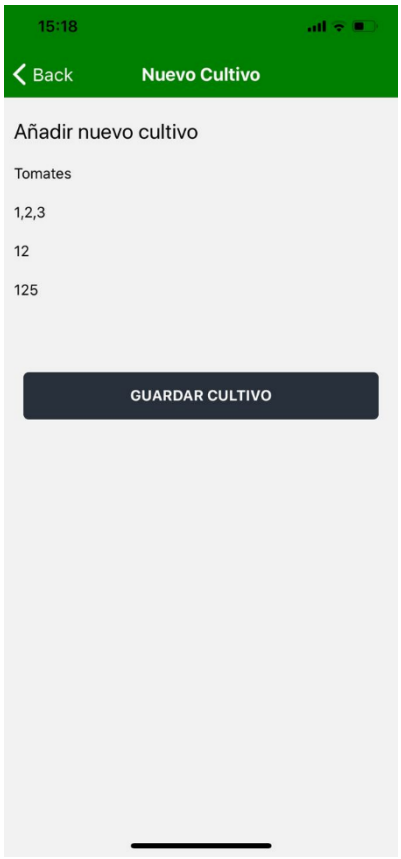

Imagen 35: NuevoCultivo.js Al pulsar editar. Fuente: elaboración propia.

3. Estadísticas: En esta pantalla se está haciendo un recopilado de la información más relevante de los riegos que tiene programado el cultivo. Se pretende con esto que el usuario pueda tener una previsión real del gasto de agua y por tanto económico que va a tener con el cultivo en cuestión. Para ello se hace una petición tipo axios.get al servidor, para recoger los parámetros de los riegos, permitiendo así obtener el total de los riegos y pudiendo ver (mediante la función diferencia) el tiempo que se está regando en cada caso y en cuantos días se produce ese riego. A su vez se recoge de la pantalla anterior el precio al que se ha marcado el m^3 de agua para poder mostrar también el gasto que se tendrá.

Por último, en la gráfica se recogen los litros que se han regado por día de forma ideal, es decir, los programados y los enfrenta a los litros que realmente se han regado, permitiendo conocer así el caudal real que se ha vertido, pudiendo comprobar la existencia de pérdidas en el sistema.

```
useEffect(() => {
    const obtenerRiegosApi = async () => {
      try {
        if (Platform.OS === "ios") {
          const resultado = await axios.get(
            `http://${ipserver}:3000/riegos?idcultivo=${idcultivo}`
          );
          guardarRiegos(resultado.data);
          guardarConsultarAPI(false);
        }
        //para android
        else {
          const resultado = await axios.get(
            `http://${ipserver}:3000/riegos?idcultivo=${idcultivo}`
          );
          guardarRiegos(resultado.data);
          guardarConsultarAPI(false);
        }
      } catch (error) {
        console.log(error);
      }
    };
    if (consultarAPI) {
      obtenerRiegosApi();
    }
  }, [consultarAPI]);
 let mintot = [0];
  let contadordias = [0];
  let x = [0];
  let numeroSectores = [0];
  let A = [0];
  let B = [0];
  let C = [0];
  let D = [0];
  let E = [0];
  let F = [0];
  let G = [0];
 riegos.forEach((element) => {
    console.log(element);
    const horaendminutos =
      parseInt(element.horaEnd.substr(0, 2)) * 60 +
      parseInt(element.horaEnd.substr(3, 5));
    const horastartminutos =
```

```
      parseInt(element.horaStart.substr(0, 2)) * 60 +
      parseInt(element.horaStart.substr(3, 5));
    const sectores = element.sectores;
    const numeroSectores = sectores.split(",").join("");
    const numero = numeroSectores.length;
    const dd = horaendminutos - horastartminutos;
    const diferencia = (horaendminutos - horastartminutos) * numero;
    contadordias[0] = 0;
    if (element.checkedL) {
      contadordias[0] = contadordias[0] + 1;
      A[0] = A[0] + 1;
    }
    if (element.checkedM) {
      contadordias[0] = contadordias[0] + 1;
      B[0] = B[0] + 1;
    }
    if (element.checkedX) {
      contadordias[0] = contadordias[0] + 1;
      C[0] = C[0] + 1;
    }
    if (element.checkedJ) {
      contadordias[0] = contadordias[0] + 1;
      D[0] = D[0] + 1;
    }
    if (element.checkedV) {
      contadordias[0] = contadordias[0] + 1;
      E[0] = E[0] + 1;
    }
    if (element.checkedS) {
      contadordias[0] = contadordias[0] + 1;
      F[0] = F[0] + 1;
    }
    if (element.checkedD) {
      contadordias[0] = contadordias[0] + 1;
      G[0] = G[0] + 1;
    }
    mintot[0] = mintot[0] + diferencia * contadordias[0];
    x[0] = x[0] + dd * contadordias[0];
    A[0] = A[0] * diferencia;
    B[0] = B[0] * diferencia;
    C[0] = C[0] * diferencia;
    D[0] = D[0] * diferencia;
    E[0] = E[0] * diferencia;
    F[0] = F[0] * diferencia;
    G[0] = G[0] * diferencia;
  });
const res = mintot[0] * caud;
  const tot = (res * (prec / 100)) / 10000;
```

Imagen 36: Graficultivo.js. Fuente: elaboración propia.

4. Gestionar Riegos: al pulsar sobre este botón, se pasa a la siguiente pantalla de InicioRiegos, a través de la propiedad onPress() del botón. Se vuelve a pasar como parámetros el estado de la consulta a la API y el id del cultivo.

```
onPress={() =>navigation.navigate("InicioRiegos", { idcultivo,
            guardarConsultarAPI,
          })}
```

InicioRiegos.js

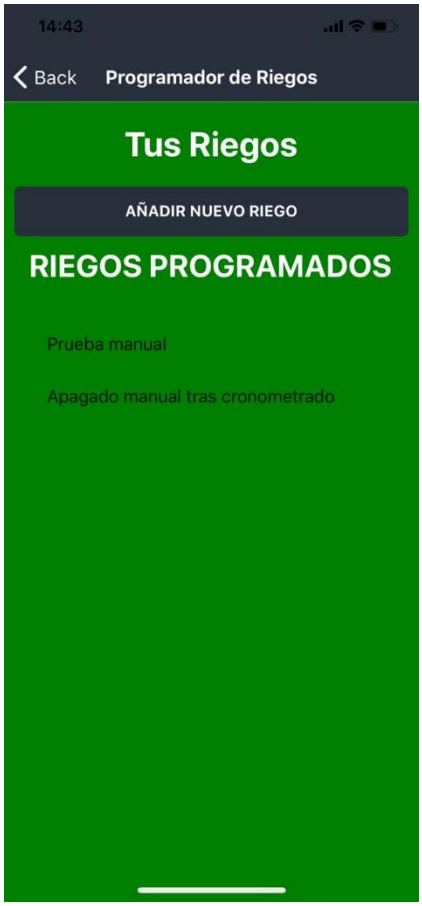

Imagen 37: InicioRiegos.js. Fuente: elaboración propia.

En las última de las pantallas anidadas, dentro de campos y de cultivos se encuentran los riegos. En esta pantalla tenemos definido un componente de texto indicando que se está en una subsección dentro de un cultivo "Tus Riegos", un botón de Añadir nuevo riego que nos conduce a la pantalla de NuevoRiego y un componente FlatList que irá listando cada uno de los riegos que vayamos añadiendo al servidor.

Para mostrar los elementos del FlatList se está haciendo de nuevo una solicitud de tipo get al servidor db.json pasando de nuevo como parámetro el id del cultivo en el que nos encontramos para evitar las consecuencias previamente aclaradas en la pantalla de InicioCultivo.

```
const obtenerRiegosApi = async () => {
      try {
          const resultado = await axios.get(
            `http://${ipserver}:3000/riegos?idcultivo=${idcultivo}`);
          guardarRiegos(resultado.data);
```

Como hemos estado programando anteriormente, cada ítem del FlatList tiene definido un OnPress

que nos conducirá a una pantalla de DetallesRiegos específica de cada ítem. Se encuentra también programada una función ternaria que, en caso de que aún no se haya añadido ningún riego, nos muestre un texto diferente en la cabecera del FlatList.

```
<Text style={styles.headline}>
        {riegos.length > 0 ? "Riegos Programados" : "Aun no hay Riegos"}
         </Text>
```

NuevoRiego.js

Imagen 38: NuevoRiego.js. Fuente: elaboración propia.

En la última de las pantallas de añadido, tenemos dos compoentes de tipos TextInput, para poder definir un nombre y los sectores que se desean regar. A continuación, nos encontramos con dos timepickers de la librería react-native-modal-datetime-picker. Estos selectores nos permiten definir una hora de inicio y de fin para los riegos, teniéndose en cuenta consideraciones tales como que no puede escogerse una hora de final más temprana que la de inicio. Por último, nos encontramos los ítems te tipo checkbox de la librería react-native-paper. El funcionamiento de estos checkbox se compone de la definición en false de las constantes (los días de la semana).

```
const [checkedM, setCheckedM] = useState(false);
```

se define el checkbox con la propiedad de pulsado onPress() y se establece que si se pulsa, cambie el valor del estado a true. Si está en true se guarda y si por el contrario está en false, se queda como desmarcado.

```
status={checkedX ? "checked" : "unchecked"}
              onPress={() => {
                setCheckedX(!checkedX);
                if (!checkedX) {
                  setCheckedX(!checkedX);
                } else {
                  setCheckedX(!checkedX);
                }}}
```

Todas las variables de los días de la semana se almacenan y se envían al servidor donde se almacena el día (lunes, por ejemplo) y el estado (true como marcado). Esto será un paso crucial para la programación autónoma del sistema de riego como se verá más adelante.

En último lugar tenemos el botón de guardar riego. Al pulsarse llama a la función guardarRiego y todos los valores anteriormente mencionados se mandan al servidor db.json a través de una petición de axio.post. Si nos encontráramos editando, se mandarían las peticiones con el axio.put, para modificar el valor existente. Al final de dicha función se encuentra un navigator que nos redirecciona a la pantalla de InicioRiego donde muestra el riego añadido en el componente FlatList (leyendo del servidor mediante el axios.get).

DetallesRiego.js

Imagen 39: DetallesRiego.js. Fuente: elaboración propia.

En esta pantalla, se muestran los detalles del riego seleccionado en el FlatList. Se muestra el nombre como título principal los sectores, las hora de inicio y fin además de los días de la semana que hemos escogido para que se riegue. Debajo de los días de la semana se puede observar un TextInput que nos pide un tiempo en minutos (de dos dígitos) precediendo al primero de los tres botones que pueden observarse. Esto sirve para determinar un número de minutos de riego si se quiere hacer de forma manual. Las funcionalidades de los botones son:

1. Encender Riego: una vez establecidos un determinado número de minutos, al pulsar la opción de encender riego, una variable definida como on con valor de estado a false, cambia su valor pasando este a valer true.

```
const [on, SetOn] = useState(false);
```

Esto hace una petición http a la dirección IP del microcontrolador. Tiene definido también una función ternaria que varía el texto del botón una vez se pulsa a "apagar texto", de tal forma que si volvemos a presionar el botón el riego se detiene mediante otra petición al microcontrolador que hace que se detenga el riego, todo esto comprobando previamente el valor de la constante on.

```
onPress={() => {
          SetOn(!on);
          if (!on) {
          console.log(minutosAR);
          const url = `http://192.168.2.92/enciende?tiempo=${minutosAR}`;
          const respuesta = fetch(url);
          console.log(respuesta);
          } else {
          const url = `http://192.168.2.92/apaga`;
          const respuesta = fetch(url);
          console.log(respuesta);}}}
```

Tal y como puede observarse, en la petición de encendido se le está pasando el número de minutos establecido "{minutosAR}". A continuación se muestra la función ternaria englobada en el botón anteriormente descrito.

```
<Text style={globalStyles.botontexto}>
              {on ? "apagar riego" : "encender riego"}</Text>
```

2. Editar riego: al pulsar sobre este botón, se redirige a la pantalla de NuevoRiego. Vuelve a realizase mediante la propiedad onPress() del botón. Se pasa como parámetros el estado de la consulta a la API y los valores que componen el riego para que puedan editarse. Al pulsar de nuevo sobre el botón de guardado, se actualizan los valores en el FlatList.

```
onPress={() =>
            navigation.navigate("NuevoRiego", {riego: route.params.item,
idcultivo, guardarConsultarAPI, })}
```

De nuevo para determinar si se está editando un cultivo o añadiendo uno nuevo, existe una función que determina si los valores del riego están vacíos (creando el riego) o si el estado de los valores está lleno (modo de edición). Si se encuentra en estado de edición, la solicitud axios que se realiza es de tipo put y en el caso contrario, mediante post.

Imagen 40: NuevoRiego.js al pulsar editar. Fuente: elaboración propia.

3. Eliminar Riego: al pulsar sobre el botón de eliminar, mediante el onPress() definido en el botón se llama a una función de mostrarConfinmación, en la cual mediante un componente Alert Si se confirma la eliminación se llama a la función elimnarRiego() que, mediante una petición axios.delete elimina el riego y deja de mostrarse en el FlatList de InicioRiego.

```
const mostrarConfimacion = () => {
    Alert.alert(
      "¿Desea eliminar el Riego?",
      "El sistema ya no regará a la hora prevista",
      [
        { text: "Si, eliminar", onPress: () => eliminarRiego() },
        { text: "Cancelar", style: "cancel" },
      ]);};
```

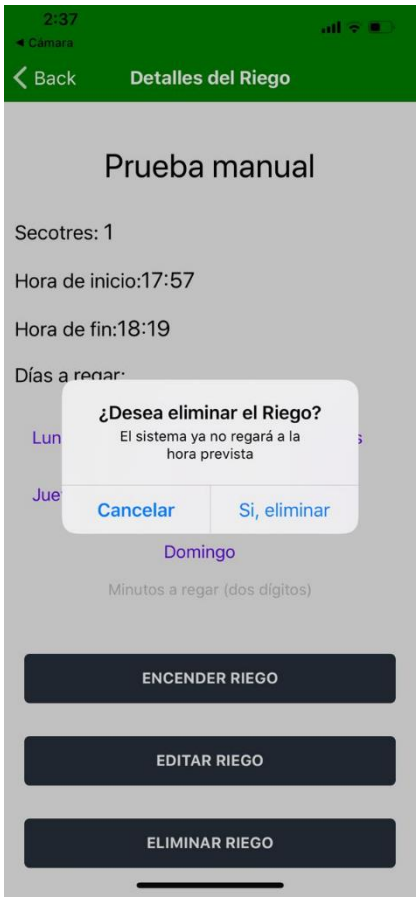

Imagen 41: DetallesRiego.js. Alerta al pulsar eliminar. Fuente: elaboración propia.

COMPARATIVA ANDROID E IOS

Para visualizar lo mencionado anteriormente, vamos a ver la importancia que tienen los componentes descritos en un código nativo y su visualización en las distintas plataformas de Android e IOS.

Aunque la mayoría de los componentes tienen una resolución similar en las plataformas, como por ejemplo los Text o TextInput, existen algunos otros como Button, Alert o Pickers que son totalmente diferentes en cuanto a visualización.

Para mantener un diseño similar en las distintas plataformas, en ocasiones se han utilizado librerías para usar componentes que también ofrece react native. Por ejemplo, para los botones. En las imágenes 42 y 43 se puede ver la pantalla de Nuevo Riego en IOS (izquierda) y Android (derecha). Los Text y TextInput se ven similares en ambas imágenes, sin embargo, en los botones de los pickers, puede apreciarse como cada plataforma aporta el suyo. De igual manera, incluso utilizando la librería de native-paper para los componentes checkbox, también se ven diferente en cada plataforma. El botón de guardado está importado de la librería de native base y es por lo que mantiene el formato para ambas plataformas.

Si pulsamos sobre los selectores de hora, también podemos observar una gran diferencia entre plataformas, al igual que si seleccionamos la opción de eliminar el riego. El mensaje de alerta se ve diferente como puede apreciarse en las imágenes 46 y 47.

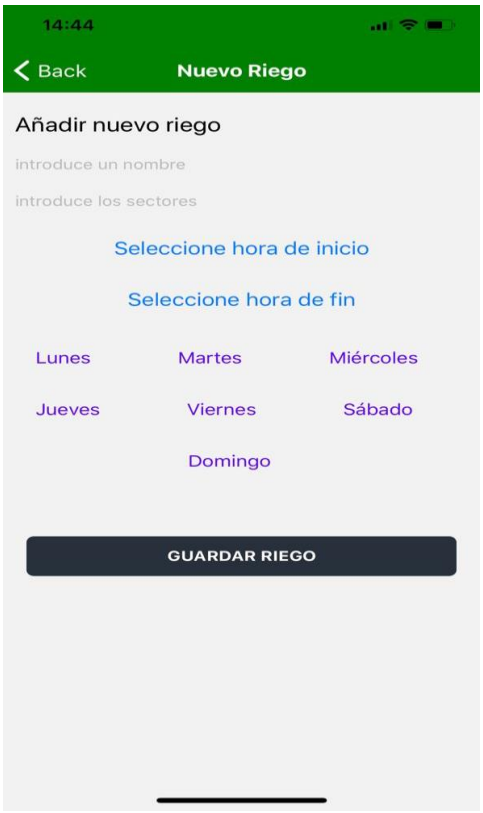

Imagen 42: Botones y checkbox vistos en terminal IOS. Fuente: elaboración propia.

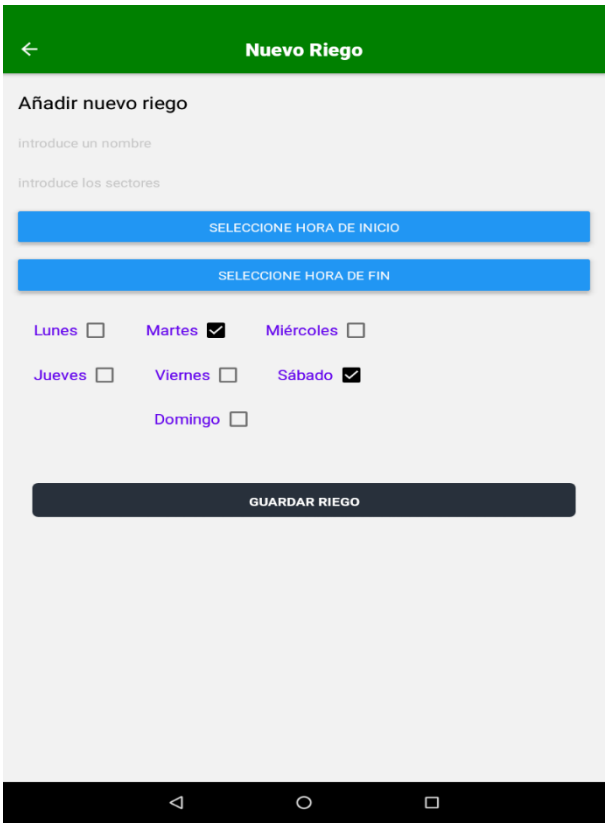

Imagen 43: Botones y checkbox vistos en terminal Android Fuente: elaboración propia.

Imagen 44: TimePicker visto en terminal IOS.

Fuente: elaboración propia.

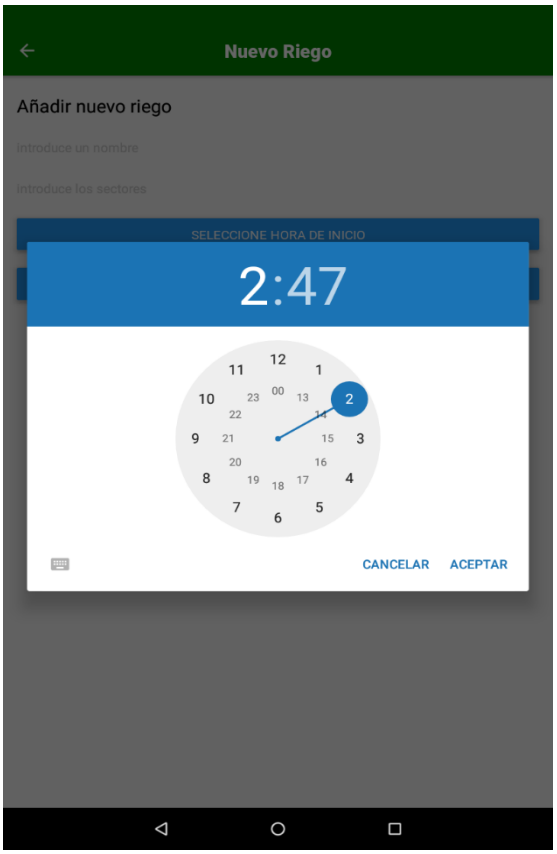

Imagen 45: TimePicker visto en terminal Android

Fuente: elaboración propia.

Imagen 46: Alert visto en terminal IOS

Fuente: elaboración propia.

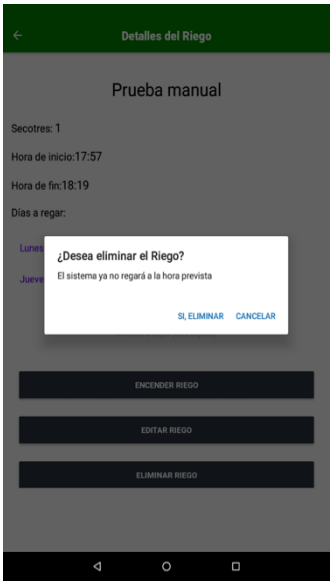

Imagen 47: Alert visto en terminal Android

Fuente: elaboración propia.

PROGRAMACIÓN DE LOS MICROCONTROLADORES

La programación del microcontrolador se ha fundamentado en dos partes destacables. Por un lado, el encendido y apagado manual y, por otra parte, la programación remota de repetición de días y horas para la semana. Es una diferenciación a tener en cuenta ya que cuando se está en modo manual, el microcontrolador actúa como servidor, mientras que, al programarse, actúa como cliente del servidor json-server. Los componentes hardware utilizaos son, un enchufe inteligente (se trata de un enchufe con capacidad de conexión de relés para el control remoto de encendido y apagado), un relé, un microcontrolador ESP8266, un motor de fuente adherido a un depósito con una manguera, por la cual se expulsa el agua succionada por el motor.

- Control manual:

Cuando nos encontramos en la pantalla de DetallesRiego para un riego completo, tal y como se explicó previamente, tenemos un botón de encendido/apagado de riego precedido por un componente TextInput, definido para establecer un tiempo determinado en minutos de riego. A través del pulsado de este botón se configura el valor del estado de la variable on y, por consecuente, se ejecuta la función definida en el if o en en else. Es fundamental rellenar dicho TextInput ya que la información proporcionada se pasará como parámetro en la petición HTTP al servidor y será el tiempo que dure el regado a menos, que se aborte la operación (detener el riego).

```
<TextInput value={minutosAR}/>
  <Button title="Empieza"
      style={styles.botonG}
      onPress={() => {
      SetOn(!on);
       if (!on) {
        const url = `http://192.168.2.92/enciende?tiempo=${minutosAR}`;
        const respuesta = fetch(url);
              console.log(respuesta);
            } else {
              const url = `http://192.168.2.92/apaga`;
        const respuesta = fetch(url);
            }}} >
          <Text style={globalStyles.botontexto}>
            {on ? "apagar riego" : "encender riego"}</Text></Button>
```

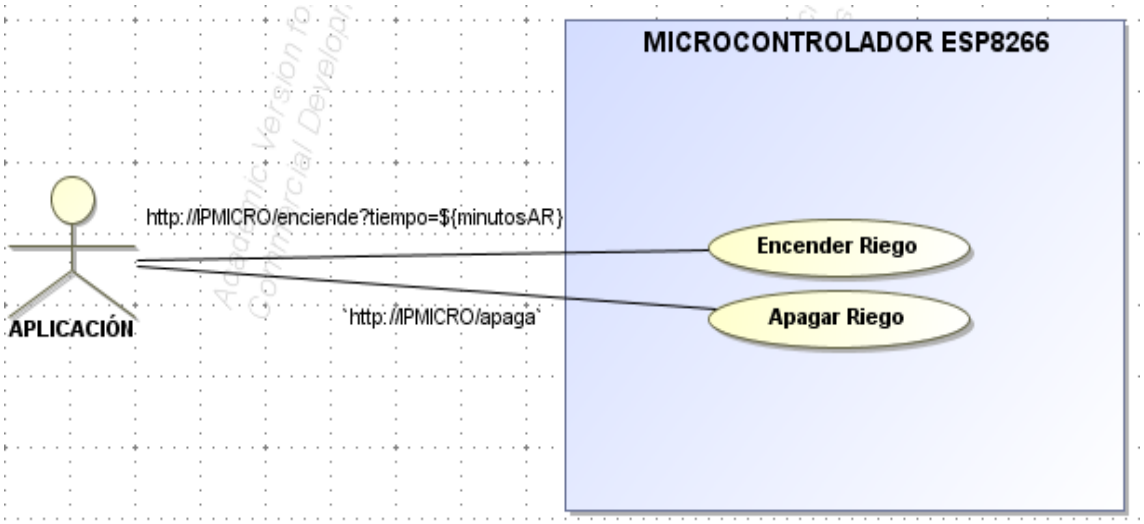

Diagrama 3: Funcionamiento Riego Manual. Fuente: Elaboración propia

Para la programación del microcontrolador:

long tiempo = 0; //PARA ALMACENAR EL TIEMPO EN MICROSEGUNDOS QUE HA INDICADO EL USUARIO PARA EL RIEGO MANUAL EN EL TEXTINPUT.

long temp_ant = 0; //PARA GUARDAR EL TIEMPO ACTUAL EN MICROSEGUNDOS.

int val; //PARA INDICAR EL VALOR (ENCENDIDO/APAGADO) DEL RELÉ.

pinMode(0, OUTPUT); //LO CONFIGURAMOS COMO PIN DE SALIDA.

digitalWrite(0, 1); //LO INICIALIZAMOS CON UN VALOR ALTO (APAGARÍA EL RELÉ, PORQUE AL CONFIGURAR EL RIEGO, ESTRÍA APAGADO).

// LEEMOS LA PRIMERA LINEA QUE NOS HA ENVIADO EL CLIENTE.

String req = client.readStringUntil('\r');

Serial.println(req);

client.flush();

if (req.indexOf("/enciende") != -1){ //SI NOS HA ENVIADO UN GET CON '/ENCIENDE'.

val = 0; //ENCENDEMOS.

tiempo = req.substring(req.length()-11,req.length()-9).toInt() * 60000; //OBTENEMOS EL TIEMPO QUE NOS HA INDICADO EL CLIENTE EN MINUTOS EN LA VARIABLE '?TIEMPO' (?TIEMPO=01, por ejemplo) Y LO GUARDAMOS EN MILISEGUNDOS.

```
  temp_ant = millis(); //COGEMOS EL TIEMPO ACTUAL EN MILISEGUNDOS.
  termina = false; //PARA QUE SE ENCIENDA.
  Serial.println("ENCIENDO DE FORMA MANUAL");
  Serial.println(tiempo);
 }
 else if (req.indexOf("/apaga") != -1){ //SI NOS HA ENVIADO UN GET CON '/APAGA'.
  val = 1; //APAGAMOS.
  termina = true; //PARA QUE SE APAGUE.
 }else { //SI ENVIA OTRA INFORMACION, LA IGNORAMOS.
  Serial.println("Informacion no valida por parte del cliente");
  client.stop();  return;  }
```

- Programación:

Para la programación será crucial el envío de datos a través de la aplicación móvil al servidor. Esta información es recogida a través de las peticiones de axio.post que se envían a crear un nuevo riego. Los valores que pueden verse dentro del riego son los que se rellenan manualmente en NuevoRiego.

Imagen 48: Programación establecida. Fuente: elaboración propia.

```
{
      "nombre": "Prueba manual",
      "sectores": "1",
      "horaStart": "17:57",
      "horaEnd": "18:19",
      "checkedL": true,
      "checkedM": false,
      "checkedX": false,
      "checkedJ": false,
      "checkedV": false,
      "checkedS": false,
      "checkedD": false,
      "idcultivo": 1,
      "id": 9
    }
```


Valores pasados al servidor db.json cuando se crea el riego


```
"horas": {
    "dia": "Thursday",
    "hora": "17:56:23,20"
  }
```


Valor que se va actualizando cada minuto mediante la ejecución de hora.cmd*

Se ha implementado un algoritmo (hora.cmd) encargado de la actualización cada minuto de la fecha y la hora. Para ello simplemente se ha creado un bucle de repetición infinita cada periodo de 60 segundos que actualiza la información del día y la almacena en otra variable del propio servidor. La que puede verse en el código de arriba "horas".

```
@echo off
set INTERVAL=60
rem get the date
rem use findstr to strip blank lines from wmic output
for /f "usebackq skip=1 tokens=1-3" %%g in (`wmic Path Win32_LocalTime Get Day^,Month^,Year ^| findstr /r /v "^$"`) do (
  set _day=00%%g
  set _month=00%%h
  set _year=%%i
  )
rem pad day and month with leading zeros
set _month=%_month:~-2%
set _day=%_day:~-2%
rem get day of the week
for /f %%k in ('powershell ^(get-date^).DayOfWeek') do (
  set _dow=%%k
  )
rem output format required is DD-MM-YYYY_weekday
:BUCLE
curl 192.168.2.71:3000/horas -H "Content-type:application/json" -X POST -d "{\"dia\":\"%_dow%\",\"hora\":\"%time%\"}"
timeout %INTERVAL%
goto BUCLE
endlocal
```

Imagen 49: Hora.cmd. Fuente: elaboración propia.

El microcontrolador por otra parte, a través de peticiones http get toma los datos de los riegos y los almacena en un fichero para poder recorrerlo. Esto permite la obtención de la hora de inicio, de fin y de los días de la semana (true) en los que debe repetirse el riego. También toma la variable horas, obteniendo el día y la hora actual, y almacenándola. Comparando todos estos valores obtenidos puede programarse los riegos de forma satisfactoria.

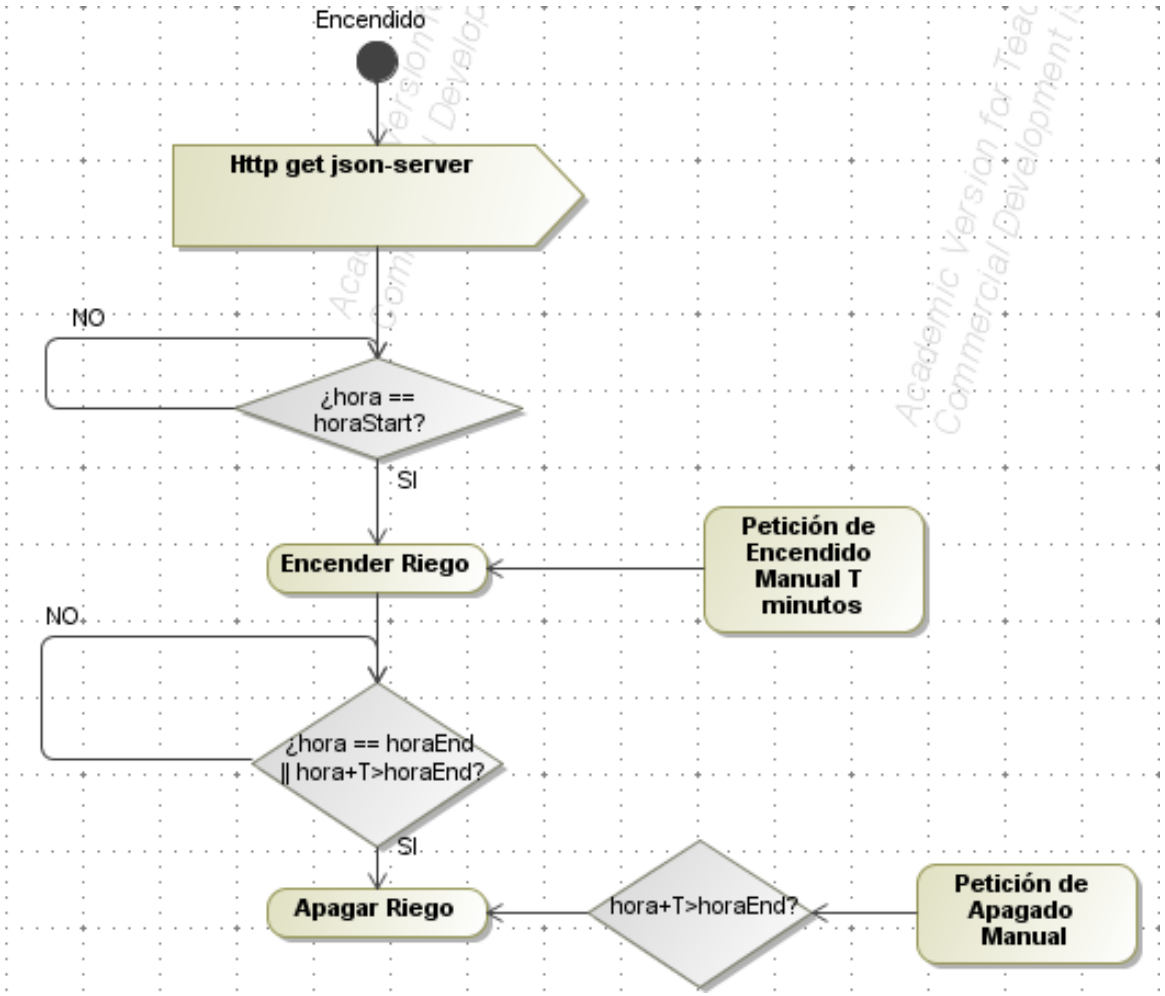

Diagrama 4: Funcionamiento de Riego Programado. Fuente: Elaboración propia

Para la programación del microcontrolador:

```
String payload = ""; //PARA RECOGER LA INFORMACION DE LOS RIEGOS.

String payload2 = ""; //PARA RECOGER LA INFORMACION DE LA HORA ACTUAL.
void loop() {
   delay(100);
   http.begin("http://192.168.2.71:3000/riegos"); //NOS CONECTAMOS AL SERVIDOR (JSON-SERVER) Y COGEMOS LOS RIEGOS PROGRAMADOS.
   int httpCode = http.GET(); //OBTENEMOS EL JSON DE LOS RIEGOS.
   if (httpCode > 0) { //SI TODO HA IDO BIEN.
     payload = http.getString(); //GUARDAMOS LA INFO. DE LOS RIEGOS EN 'PAYLOAD'.
   }
   else{ //SI HA IDO MAL.
     Serial.println("Error:");    }
   http.end(); //FINALIZAMOS LA CONEXION AL SERVIDOR JSON-SERVER.


bool acceso = false; //PARA COMPROBAR SI SE TIENE QUE ENCENDER EL RIEGO PROGRAMADO (==TRUE, SE TIENE QUE ENCENDER).
 for(int i = 0; i<doc.size();i++){ //RECORREMOS EL DOCUMENTO.
   int id = doc[i]["id"]; //OBTENEMOS LOS ID DE LOS RIEGOS .
   String horaStart = doc[i]["horaStart"];
   String horaEnd = doc[i]["horaEnd"];
   bool checkedL = doc[i]["checkedL"];
   bool checkedM = doc[i]["checkedM"];
   bool checkedX = doc[i]["checkedX"];
   bool checkedJ = doc[i]["checkedJ"];
   bool checkedV = doc[i]["checkedV"];
   bool checkedS = doc[i]["checkedS"];
   bool checkedD = doc[i]["checkedD"];
  //COMPROBAR SI EN LA HORA Y DIA ACTUAL DEBERIA DE ESTAR ENCENDIDO.SI ES QUE SI, ENTRAR EN UN BUCLE DE ENCENDIDO CON EL TIEMPO PROGRAMADO.
   http.begin("http://192.168.2.71:3000/horas"); //OBTENEMOS LA HORA Y EL DIA DE LA SEMANA DEL JSON-SERVER.
   int httpCode = http.GET();
   if (httpCode > 0) { //SI TODO HA IDO BIEN.
     payload2 = http.getString(); //GUARDAMOS LA INFORMACIÓN DE LA HORA Y EL DIA DE LA SEMANA EN 'PAYLOAD2' }.
  else{ //SI HA IDO ALGO MAL.
```

```
    Serial.println("Error:");
  }
  http.end(); //FINALIZAMOS LA CONEXION AL JSON-SERVER.
String dia = doc2["dia"]; //OBTENEMOS EL DIA DE LA SEMANA.
  String horario = doc2["hora"]; //OBTENEMOS LA HORA ACTUAL.
  horario = horario.substring(0,5); //NOS QUEDAMOS CON 'HORA:MINUTOS' (HH:MM).
if(ComparaHorario(horario, horaStart, horaEnd)){ //COMPROBAMOS SI LA HORA ACTUAL ESTA DENTRO DE UN HORARIO DE ALGUN RIEGO.
    if(checkedL and dia.equals("Monday")){ //SI EL RIEGO ESTA PROGRAMADO PARA LUNES, Y ESTAMOS A LUNES.
      Enciendete(horaStart,horaEnd);
      acceso = true;
    } . . . //PARA TODOS LOS DIAS DE LA SEMANA.
if(!acceso and termina){ //SI NO HAY RIEGO PROGRAMADO Y NO HAY RIEGO MANUAL.
  Apagate();
 }
```

A parte del riego programado y el encendido/apagado manual, se han contemplado una serie de casos que podrían darse dentro de la aplicación, llámense excepciones.

Se ha tenido en cuenta el caso de la prolongación de un regado en marcha mediante programación manual, es decir, si a falta de X minutos para terminar se programa un riego manualmente de un tiempo T que sobrepase la hora programada de fin, el tiempo establecido manualmente se regará.

También se ha contemplado el caso de que se intente apagar un regado que esté en ejecución de forma manual, en este caso, al intentar reducir el tiempo final establecido, se haga caso omiso a la orden manual.

Por último, se ha considerado habilitar el apagado manual de un riego de tiempo establecido manualmente después de una programación. Como la hora de fin programada es menor a la hora actual, si se puede cancelar el riego manualmente.

PRUEBAS DE FUNCIONAMIENTO

A modo aclaratorio de la explicación y el código escrito en el entorno de Arduino para programar el microcontrolador ESP8266 que coordine y gestione los riegos, se va a proporcionar a continuación una serie de enlaces cuyo contenido son videos de elaboración propia que reproducen las pruebas (descritas en su título) que se han llevado a cabo para comprobar el correcto funcionamiento de enlazado entre el software y el hardware. Puede verse como lo escrito en la aplicación (cliente) es guardado en el servidor y cómo el microcontrolador (cliente) es capaz de leer los datos del servidor proporcionados por la aplicación de usuario.

- Encendido y apagado manual:

  https://youtu.be/PKCnyjrhSwA

- Temporización manual de un minuto:

  https://youtu.be/stQ-NJW-4rw

- Riego programado, actuación autónoma:

  https://youtu.be/0hyr--YJ0FY

- Excepciones contempladas:

  1. Durante un riego corriendo con hora de fin programado, mediante activación manual prolongar el regado el tiempo establecido

     https://youtu.be/E5o4dOqbTn0

  2. Prevalencia de regado al intentar apagarse manualmente un riego durante un intervalo establecido:

https://youtu.be/_fEpD7net_4

3. Terminado de riego manualmente establecido durante un programado (cuando ha sobrepasado la hora de terminado)

https://youtu.be/2vafY4hFzGw

LÍNEA DE DESARROLLO

Con carácter a futuro de una posible implementación real en un sistema de riego, pueden resultar de interés, pero que, sin embargo, para este proyecto, no se han tenido en consideración.

1. A modo de interacción con el usuario, puede resultar de interés desarrollar una implementación de notificaciones que avisen al usuario del inicio y el fin de los riegos, así como de qué riego y en qué campo se está llevando a cabo.
2. Se han implementado microcontroladores ESP8266 ya que poseen la capacidad de control telemático. Pueden añadirse dentro de un sistema real sensores de humedad que se conecten al servidor autónomamente y se active el riego hasta llegar a un nivel de humedad establecido sin la necesidad de interacción del usuario.
3. Si se rigiera el sistema de riego por un depósito, también cabría la posibilidad de usar sensores que fueran capaces de detectar el nivel de agua que tiene el depósito y, en el caso de necesitar rellenarse, avisara al usuario.
4. Podría estudiarse el uso de electroválvulas u otros dispositivos que controlaran el paso del agua. De esta forma podría utilizarse de manera real el caudal de riego, permitiendo así la existencia de regado por goteo. No obstante, el uso de estos dispositivos requeriría otro tipo de controladores que tuvieran algún pin de control de potencia (la Raspberry pi 3, por ejemplo)
5. Podría programarse mediante alguna API de climatología que los riegos programados si está lloviendo en ese momento, esta programación no se llevara a cabo.
6. Exportar la aplicación a un servidor que permitiera la extracción del SDK para publicarla en el market de las distintas plataformas. Esto conllevaría la implementación de una base de datos.
7. Integrar el proyecto en la nube con IFTTT, que según su página oficial (https://ifttt.com/) es la forma gratuita de hacer que todas las aplicaciones y dispositivos se comuniquen entre sí. Permitiendo entre otras cosas, la asignación de IP de los dispositivos de manera no manual.

BIBLIOGRAFÍA

- NAVARRETE, Toni. El lenguaje JavaScript. Argentina, 2006. Recuperado de: http://www.academia.edu/download/54012715/javascript.pdf

- PUIG, Jordi Collell. CSS3 y Javascript avanzado, 2013. Diponible en https://openlibra.com/es/book/download/css3-y-javascript-avanzado

- MDN web docs. JavaScript, 2020. Recuperado de: https://developer.mozilla.org/es/docs/Web/JavaScript

- REACT JS. ¿Qué es React?, 2020. Recuperado de: https://es.reactjs.org

- REACT NATIVE. Reart Native. Learn once, write anywhere, 2020. Recuperado de: https://reactnative.dev/

- NODE JS. Acerca de Node.js, 2020. Recuperado de https://nodejs.org/es/about/

- NPM. About npm, 2020. Recuperado de https://www.npmjs.com/about

- YAGÜE, José Luis Fuentes. *Técnicas de riego*. Ministerio de Agricultura, Pesca y Alimentación, 2003. Recuerpado de https://books.google.es/books?hl=es&lr=&id=P-FByIgVfNoC&oi=fnd&pg=PA5&dq=t%C3%A9cnicas+de+riego&ots=kk_hZXC0eI&sig=CxP1INcfGKIiTvrFtd56hA366UA#v=onepage&q=t%C3%A9cnicas%20de%20riego&f=false

- MY GARDEN. Gardening information. 2020. Recuperado de https://www.mygarden.org/

- GARDEN ANSWERS. About us. 2020. Recuperado de http://www.gardenanswers.com/

- GLOBALCAMPO. Inicio. 2020. Recuperado de https://globalcampo.es/

- CULTIVAPP. Bienvenido a CultivAPP. 2020. Recuperado de https://www.cultivapp.com/

- CAMPOGEST. Campogest. 2020. Recuperado de https://www.campogest.com

- IRIEGO. Danuve & RiegoApp. 2020. Recuperado de https://www.iriego.es

- MINISTERIO DE AGRICULTURA, PESCA Y ALIMENTACIÓN. SiarApp. 2020. Recuperado de http://eportal.mapa.gob.es/websiar/Inicio.aspx

- ROSLI, Rafhanah Shazwani; HABAEBI, Mohamed Hadi; ISLAM, Md Rafiqul. Characteristic analysis of received signal strength indicator from esp8266 wifi transceiver module. En 2018 7th International Conference on Computer and Communication Engineering (ICCCE). IEEE, 2018. p. 504-507.

- ARJADI, R. Harry, et al. RSSI Comparison of ESP8266 Modules. En 2018 Electrical Power, Electronics, Communications, Controls and Informatics Seminar (EECCIS). IEEE, 2018. p. 150-153.

- IFTTT . Ifttt 2020 recuperado de https://ifttt.com/

- NODE.JS. Node.js 2020 Recuperado de https://nodejs.org/es/