

# PROYECTO FIN DE CICLO FORMATIVO

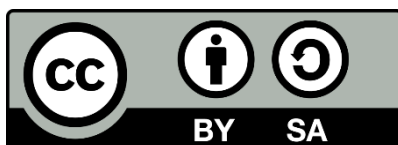
## TÉCNICO SUPERIOR EN DESARROLLO DE APLICACIONES MULTIPLATAFORMA

**AGRUINO**

AUTOR(ES)

*D. Antonio Carmona López*

TUTOR



## ÍNDICE GENERAL

<b>2. ANÁLISIS DEL PROYECTO.....</b>	<b>7</b>
2.1. BASES TEÓRICAS.....	7
2.2. SERVICIOS UTILIZADOS.....	7
2.3. HERRAMIENTAS UTILIZADAS.....	7
2.4. LENGUAJES DE PROGRAMACIÓN UTILIZADOS.....	8
2.5. OBJETIVOS.....	8
2.6. ANÁLISIS REQUISITOS.....	8
2.7. PLANIFICACIÓN.....	9
<b>3. DISEÑO DEL PROYECTO.....</b>	<b>10</b>
3.1. DISEÑO B.D (NOSQL).....	10
3.2. COLECCIONES BD DEL PROYECTO:.....	10
3.3. CLASES PROYECTO.....	11
3.4. DIAGRAMA DE FLUJO DEL PROYECTO.....	13
3.5. DISEÑO INICIAL DE LAS INTERFACES DE LOS CLIENTES WEB Y ANDROID.....	14
3.5.1. <i>Cliente web:</i> .....	14
3.5.2. <i>Cliente android:</i> .....	15
3.6. ESQUEMAS MONTAJE SENSORES ARDUINO.....	17
3.6.1. <i>Sensor de humedad:</i> .....	17
3.6.2. <i>Sensor conductividad del agua:</i> .....	18
3.6.3. <i>Sensor turbidez del agua:</i> .....	19
3.6.4. <i>Sensor ph del agua:</i> .....	20
3.6.5. <i>Sensor temperatura agua:</i> .....	21
3.7. CASOS DE USO.....	22
3.7.1. <i>Autenticación:</i> .....	22
3.7.2. <i>Gestión de usuarios (Solo cliente web):</i> .....	23
3.7.3. <i>Recoger valores:</i> .....	24
3.7.4. <i>Consultar valores:</i> .....	25
3.7.5. <i>Caso uso enviar email (solo cliente web):</i> .....	26
<b>4. DESARROLLO DEL PRODUCTO SOFTWARE.....</b>	<b>28</b>
4.1. PROGRAMACIÓN.....	28
4.1.1. <i>Servidor de datos (Arduino):</i> .....	28
4.1.2. <i>Respaldo de datos en firestore (Nodejs):</i> .....	28
4.1.3. <i>Reglas Firestone:</i> .....	29
4.1.4. <i>Características propias del cliente Android:</i> .....	30
4.1.5. <i>Características propias de cliente Web (Angular 8):</i> .....	32
4.2. PRUEBAS:.....	34

4.2.1 Captura datos que Arduino recolecta de sus diferentes sensores a el acoplados:.....	34
4.2.2. Respalda datos en BD:.....	34
4.2.3. Recoger datos por parte de clientes:.....	34
<b>5. MANUAL DE USUARIO.....</b>	<b>36</b>
5.1. APLICACIÓN WEB:.....	36
5.1.1. Despliegue:.....	36
5.1.2. Funcionalidad:.....	37
5.2. CLIENTE ANDROID:.....	40
5.2.1. Generar paquete distribución:.....	40
5.2.2. Funcionalidad:.....	40
<b>6. CONCLUSIONES Y VÍAS FUTURAS.....</b>	<b>42</b>
<b>7. REFERENCIAS Y BIBLIOGRAFÍA.....</b>	<b>46</b>

## 1. INTRODUCCIÓN.

Dada la creciente necesidad de agua de calidad para el riego, en particular en esta zona del levante español, pero extensible a cualquier latitud. Desde este proyecto se pretende dar una respuesta a dicha necesidad, optimizando su uso y, dado su bajo coste es ideal para una pequeña explotación agrícola ya que para su mantenimiento no se requieren muy pocos conocimientos previos. Pero siempre teniendo en cuenta que los medios utilizados para la medición no son más que los que se usarían en un prototipo.

Todo lo anteriormente comentado se ha hecho bajo la premisa de la viabilidad económica del proyecto y, aunque hay que poner el proyecto en el contexto de que se encuentra en un estado germinal, por lo que para su implantación quizá se requieran de más medios tanto, a nivel técnico como económico mayores.

La parte cliente (*Angular 8 y Android*) que nos hará las veces de monitoreo desde cualquier lugar, si bien es prescindible, tampoco se perderían al 100% la parte de motorización, tan solo habrá que mandar la salida a un LCD, pero si se opta por utilizar el cliente para hacer un seguimiento continuo desde cualquier lugar de los parámetros del cultivo, el único factor limitante será el acceso a internet, bien vía 4G, ¿5G?, inalámbrico, alámbrico, radio, etc. ...Con objeto de minimizar esta limitación se ha configurado *Firestore* para utilizar persistencia de datos.

También se puede optar por que ambas funcionalidades al mismo tiempo, asegurándonos que, si por cualquier causa internet se interrumpe, el monitoreo de la explotación agrícola no se verá interrumpido.

Por experiencias previas. instalar un LCD en *Arduino* es relativamente sencillo, dependerá del *modelo de LCD*, pero el *modelo básico probado ha sido fácil de instalar y programar para que Arduino* le mande la información recogida.

Por otra parte, los clientes web y *Android* que, nos mostrara en todo momento los valores recogidos en nuestro cultivo, pueden ser una buena herramienta para intentar adivinar la evolución de nuestra explotación agrícola.

\* *Arduino*, placa de extensión y todos los sensores utilizados, no son más que herramientas de modelaje.



## 2. ANÁLISIS DEL PROYECTO.

### 2.1. BASES TEÓRICAS.

En primer lugar, se tomarán distintos parámetros relativos a la humedad del suelo y calidad del agua, sita en un tanque de agua, todo ello se realizará con la ayuda de *Arduino*, con unos sensores que este llevará instalados y, en función de los datos provenientes de las lecturas, activará o desactivará automáticamente el riego, mediante la activación o desactivación de una bomba de agua.

En un segundo estadio, y mediante la plataforma *NodeJS*, se ejecutará un script desarrollado con *Javascript*, donde se recogerán del puerto serie los datos provenientes de las diferentes lecturas que realizara *Arduino* y se guardarán en *Firestore*, done este script cada vez que se active o desactive el riego, se enviara un mensaje a *Telegram*, mediante un script desarrollado con *Bash* (Shell scripting/Linux).

Una vez almacenados los datos en *Firestore*, los clientes desarrollados, uno con el framework de *Typescript* angular y otro con *Android Studio*, se procederá a la lectura de los datos por parte de este último.

### 2.2. Servicios utilizados.

*Firestore* (base de datos en tiempo real).

*Telegram*.

### 2.3. Herramientas utilizadas.

*Arduino uno*.

*Android Studio*.

*Visual paradigm*.

*Pencil Project* (modelado interfaces).

*GIMP* (edición grafica)

*Fritzing*, modelar circuitos electrónicos.

*VsCodium*.

*Ide arduino.*

*NodeJS 10 (framework javascript).*

*Angular 8 (framework typescript).*

*BootStrap 4.*

## **2.4. Lenguajes de programación utilizados.**

*Propio de arduino.*

*Java.*

*Shell script (scripting de Linux).*

*Javascript/JQUERY.*

*Typescript.*

## **2.5. Objetivos.**

1. Capturar por parte de *Arduino* las señales que desde los sensores acoplados a el, legan.
2. Respalidar datos en BD.
3. Consumir datos respaldados en BD por parte de clientes.

## **2.6. Análisis requisitos**

1. Capturar, por parte de *Arduino*, los eventos físicos recogidos por los distintos sensores a el acoplados y que este los interprete a valores numéricos, para lo cual se requerirá la programación de los distintos sensores. Una vez capturados e interpretados los valores. Una vez interpretados, se evaluarán para enviar señal a la bomba de riego, para que se encienda o apague dependiendo de si se han rebasado los umbrales previamente establecidos. Tras lo cual se procederá a confeccionar un string separado por espacios con los distintos valores recogidos e interpretados, para que *Arduino* la envíen por el puerto serie.

2. Recoger string que *Arduino* envió al puerto serie, por medio de un script desarrollado con la plataforma *NodeJS*, separar los datos en variables, evaluarlos para enviar mensaje al bot de Telegram con información del estado del riego y respaldarlos en *Firestore*, casteados a *doublé*.
3. Una vez respaldados los datos en *Firestore*, consumirlos con clientes *Android* y web (*Angular 8*).

## 2.7. Planificación

1. Capturar las señales que nos envían los distintos sensores instalados en *Arduino*, procesarlos para traducirlos a un valor numérico y finalmente recoger las diferentes lecturas construir una cadena de texto con todos los valores y enviarla al puerto serie.
2. Recoger con un script en *Nodejs* cadena de caracteres con los datos provenientes de *Arduino*., separarlos convertirlos a valores numéricos y subirlos a *Firestore*.
3. Una vez almacenados los diferentes valores, se procederá a su lectura por parte de los clientes.



### 3. DISEÑO DEL PROYECTO.

#### 3.1. Diseño B.D (noSQL)

En el diseño de la BD construiremos tres colecciones para albergar los datos que arduino nos enviara a través del puerto serie:

1. Colección values.
2. Colección values\_history.
3. Colección values\_log.

Y una cuarta colección a modo de contenedor de información de los usuarios del sistema.

#### 3.2. Colecciones BD del proyecto:

users	displayName: "Admin"	users	
values	email: "antocarmona@gmail.com"	values	conductivity: 0.14
values_history	emailVerified: true	values_history	moisture: 4
values_log	photoURL: "https://firebasestorage.googleapis.com/v0/b/agruino.appspot.com/o/uploads%2Fprofile_ip4iuwqy8t%2Ftoken-f0f8f0d8-65a5-4333-bbd3-ed7dae8b2d3f"	values_log	ph: 26.67
			temp: -23.85
			turbidity: 4
users	conductivity: 4.662	users	conductivity: 0.25
values	moisture: 104	values	date: 1356998400000
values_history	ph: 70	values_history	dateString: "01/01/2013"
values_log	temp: -27.700000000000003	values_log	moisture: 5
	turbidity: 8		ph: 25
			temp: 59.23
			time: "8:00"
			turbidity: 2

En la colección user se creará un registro con diversa información acerca del usuario registrado para, que este pueda consultar y modificar alguna información de su perfil y en el caso de los usuarios con rol admin podrán, la lista de todos los usuarios del sistema, para enviarle un email o borrarlo.

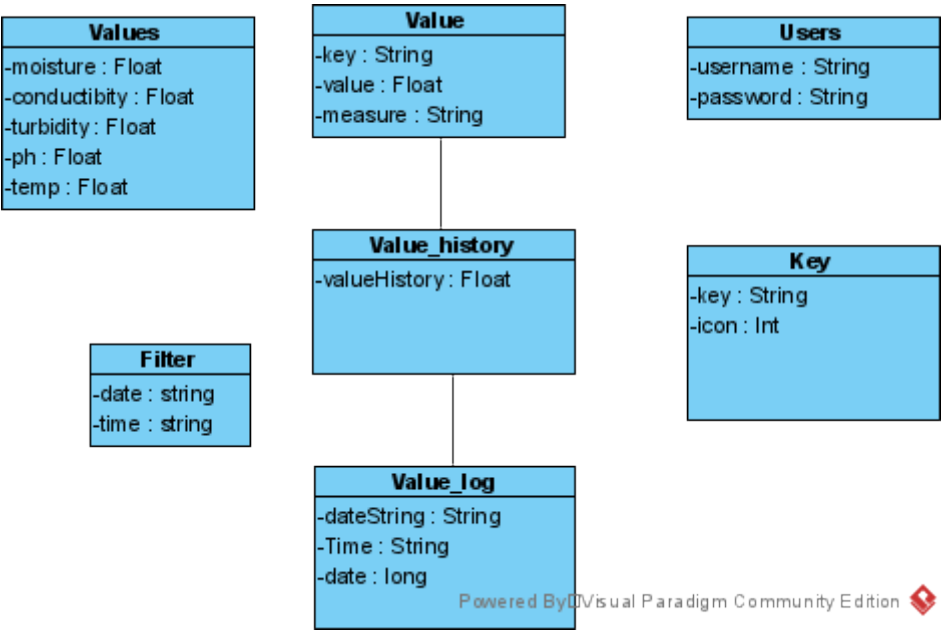
Las colecciones values, values-history y values-log, almacenaran datos provenientes de las diferentes lecturas, comportándose de distinta manera en función a su función:

- ✓ Values: colección destinada a albergar la última lectura recibida proveniente de arduino.
- ✓ Values\_history: colección destinada a albergar el sumatorio de las distintas lecturas recibidas de *Arduino*, para más tarde calcular media.
- ✓ Values\_log: colección que alojara documentos con lecturas realizadas, con motivo de más tarde utilizarlas a modo de auditoria.

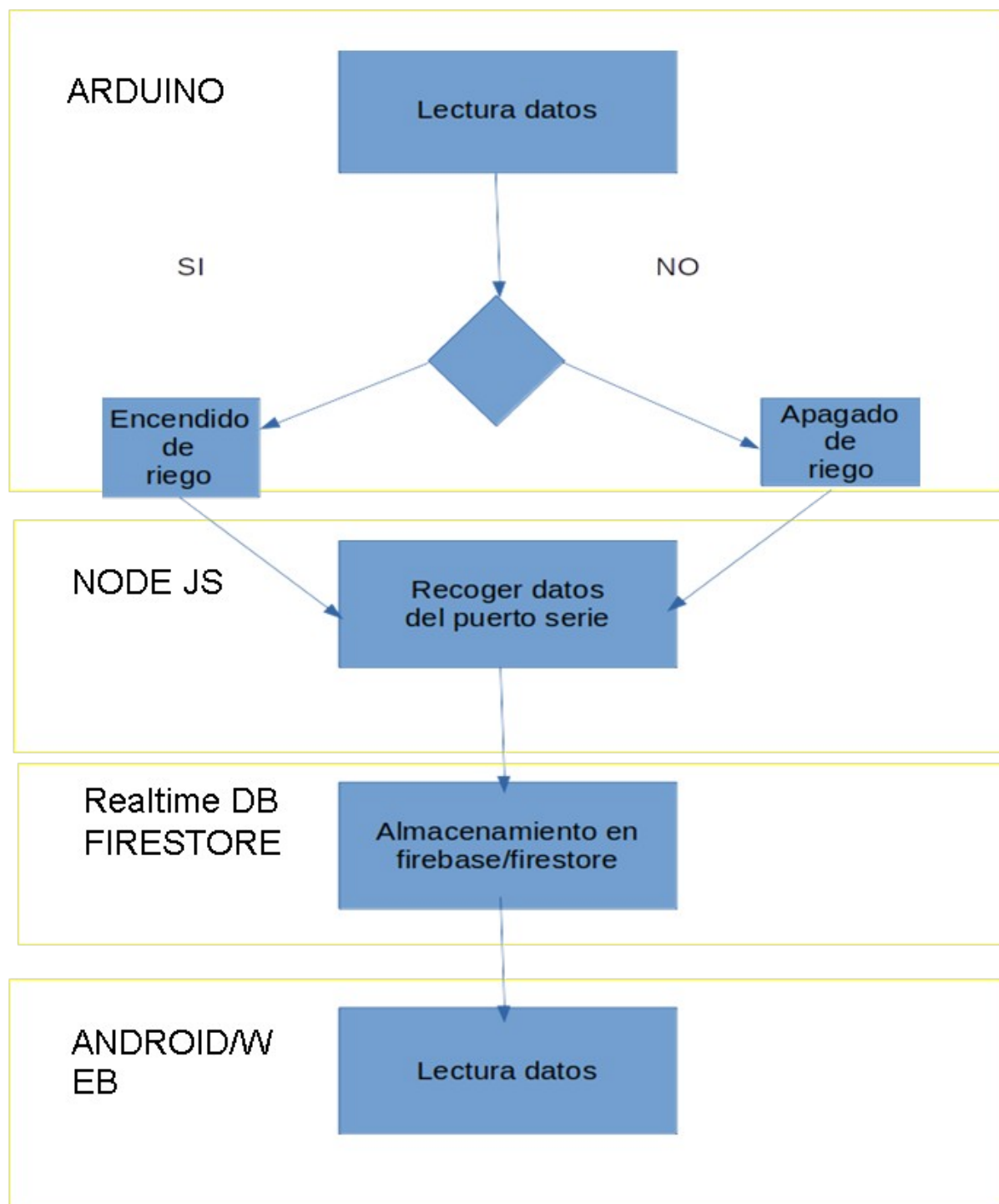
### 3.3. Clases proyecto

Estas clases solamente se han utilizado en el cliente *Android*, ya que el cliente web, que de forma indirecta maneja *Javascript*, resulta más cómoda una programación basada en funciones.

- ✓ Clase user: Clase destinada a contener los string username y password, para utilizar dicha información en el método que nos autenticara en el sistema.
- ✓ La clase key es la clase que utilizaremos para pasarle información al fragmento de la lectura que debe mostrar.
- ✓ La clase value será la destinada a albergar información de la lectura y que mostraremos en los textbox.
- ✓ La clase Value\_history, hereda de la anterior y, a a la información de value, se le añade su valor acumulado a lo largo del año.
- ✓ La clase value\_log hereda de value\_history y albergara lecturas almacenadas en la colección values\_log.
- ✓ La clase value albergara la última lectura recibida de *Arduino* y, de esta forma evitar accesos recurrentes a BD.
- ✓ La clase filter contendrá la información por la que deseamos filtrar las values\_log.

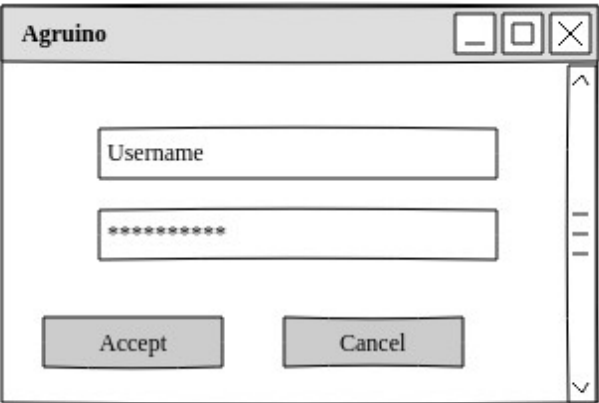


### 3.4. Diagrama de flujo del proyecto



**3.5. Diseño inicial de las interfaces de los clientes web y android.**

**3.5.1. Cliente web:**



A login window titled "Agruino" with a standard window control bar (minimize, maximize, close). It contains two input fields: "Username" and a password field represented by "\*\*\*\*\*". Below the fields are two buttons: "Accept" and "Cancel". A vertical scrollbar is on the right side.

El dashboard tendrá diferentes opciones dependiendo del rol del usuario logeado:



\* dashboard admin



\* dashboard user

### 3.5.2. Cliente android:



Si en la pantalla de login nos logueamos correctamente, la app nos presentara un RV donde consultar los valores de humedad (ruta 1), agua (ruta 2) y valores históricos de lecturas(ruta 3)

Si pinchamos sobre este icono, nos dirigiremos al visualizar lectura de humedad

\* RUTA 1



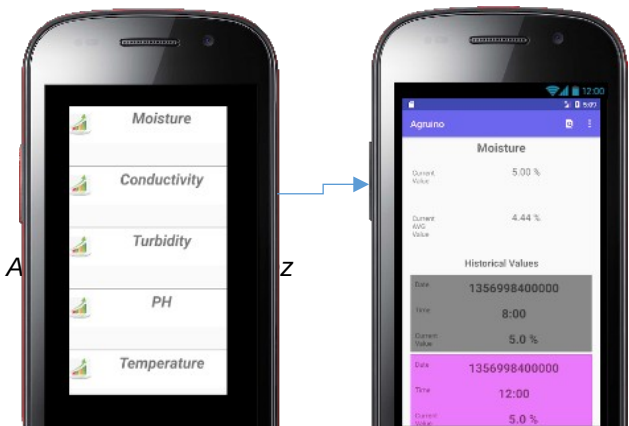
Si pinchamos sobre este otro icono, no dirigiremos a otro fragmento donde, se nos presentará un recyclerView con las distintas opciones.

\* RUTA 2



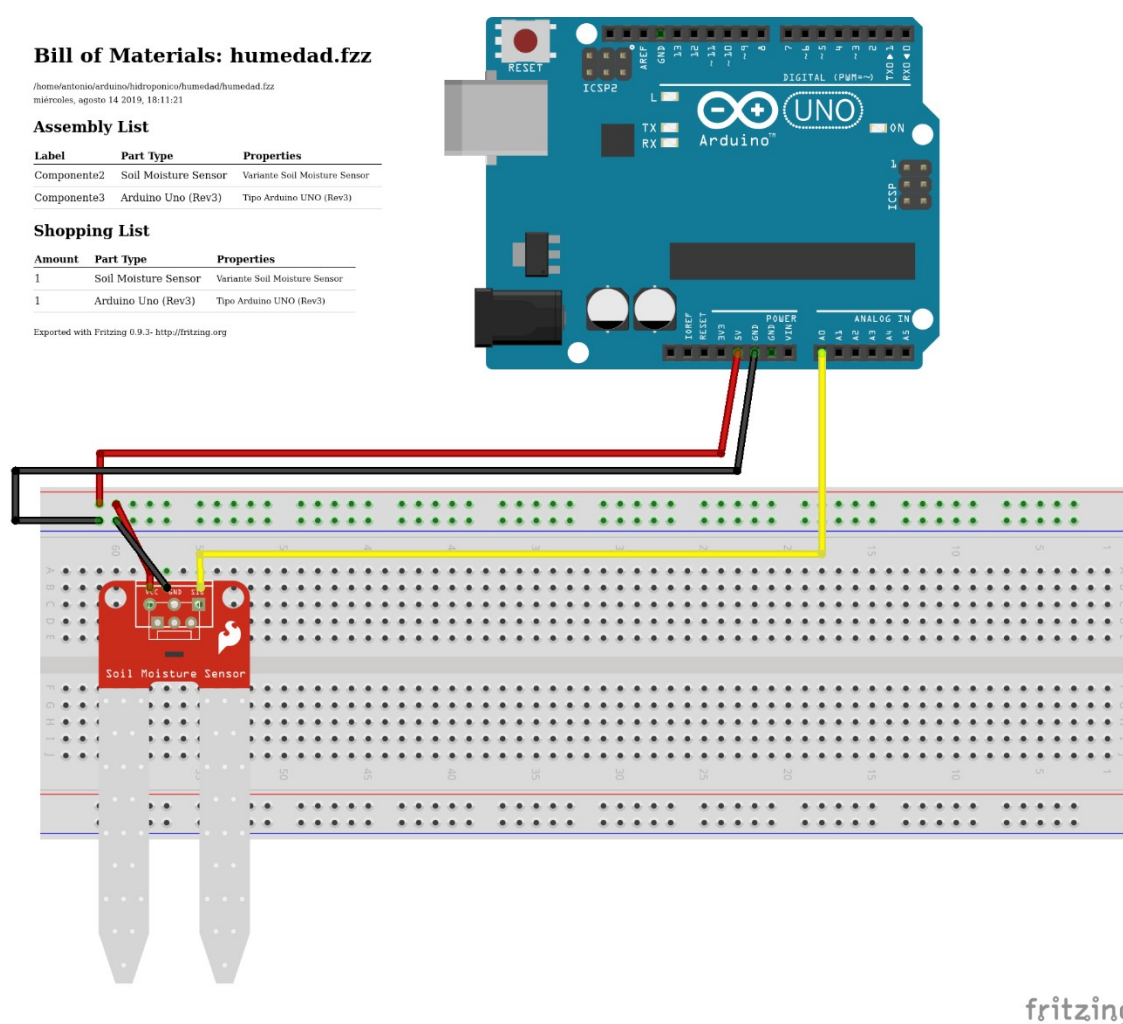
Si pinchamos sobre este tercer CV, la app nos dirigirá a otro RV con un CV para cada valor y, poder consultar su valor medio anual.

\* RUTA 3



## 3.6. Esquemas montaje sensores arduino

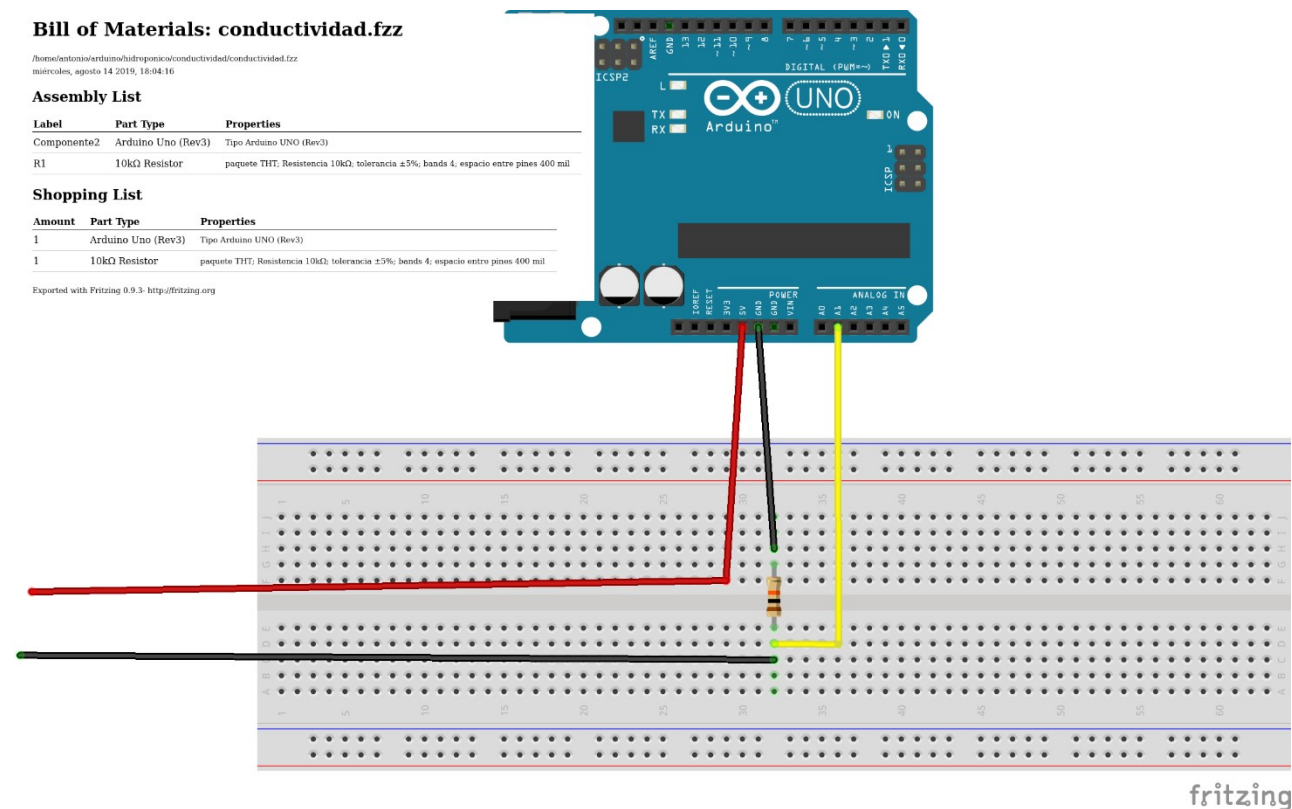
### 3.6.1. Sensor de humedad:



- ✓ Este sensor no requiere, para su correcto funcionamiento, de resistencia alguna ya que lleva incorporado un potenciómetro que mantendrá siempre un estado alto cuando no se está transmitiendo y así no solapar lecturas



### 3.6.2. Sensor conductividad del agua:



✓ Como se puede apreciar en el esquema de montaje, este dispositivo si necesita de una resistencia de 10 kΩ para mantener un estado alto y dado que el precio de este sensor es alto para una simple maqueta, he decidido [fabricarlo](#), para lo cual no han sido necesario una ingente cantidad de conocimientos, tan solo con física básica y un poco de pericia.

3.6.3. Sensor turbidez del agua:

**Bill of Materials: turbidez.fzz**

Assembly List

Label	Part Type	Properties
Componente 1	Arduino Uno (Rev3)	Typo Arduino UNO (Rev3)
Turbidity Sensor	KS0414 Keyestudio Turbidity Sensor V1.0	Range de 0%-3.5% (0-4550NTU), con un error de ±0.5%

Shopping List

Amount	Part Type	Properties
1	Arduino Uno (Rev3)	Typo Arduino UNO (Rev3)
1	Turbidity Sensor	KS0414 Keyestudio Turbidity Sensor V1.0 Range de 0%-3.5% (0-4550NTU), con un error de ±0.5%

Exported with Fritzing 0.9.3: <http://fritzing.org>

✓ Este sensor se sumergirá en el tanque de agua. Una vez sumergido este proyectará un haz coherente de luz y en función de la dispersión de este haz de luz nos dará una lectura entre 0% y 3.5%, siendo 0 agua limpia y 3.5 agua totalmente turbia, dándonos una lectura de 4550 NTU (Unidades Nefelométricas de turbidez) y, no apta ni para el riego y mucho menos para el consumo humano.

3.6.4. Sensor ph del agua:

**Bill of Materials: Ph meter Sketch.fzz**

**Assembly List**

Label	Part Type	Properties
Componente1	Arduino Uno (Rev3)	Tipo Arduino UNO (Rev3)
S1	PH meter	Alfileres 3; espacio entre pines 0.1in (2.54mm); Variante variant 1; paquete THT; row single

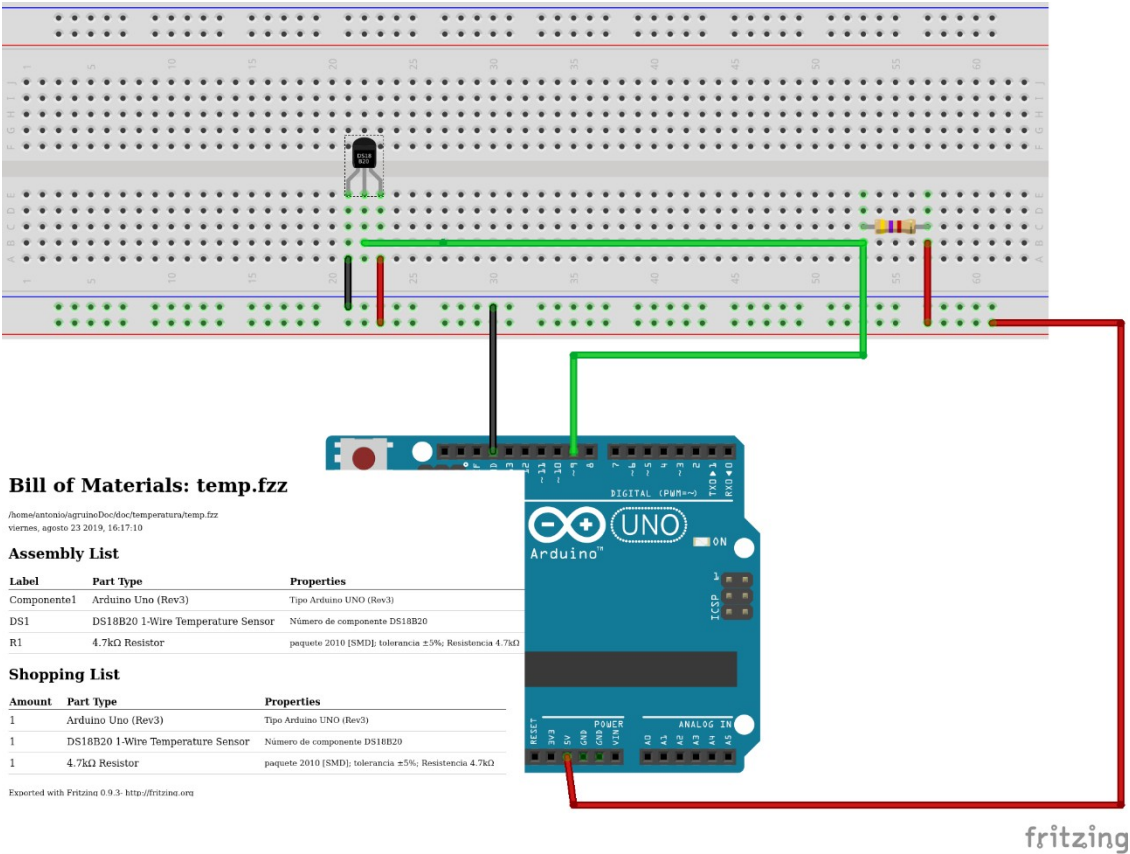
**Shopping List**

Amount	Part Type	Properties
1	Arduino Uno (Rev3)	Tipo Arduino UNO (Rev3)
1	PH meter	Alfileres 3; espacio entre pines 0.1in (2.54mm); Variante variant 1; paquete THT; row single

Exported with Fritzing 0.9.3 - <http://fritzing.org>

✓ Este sensor se sumergirá en el tanque de agua. Una vez sumergido nos indicara la acidez del agua, midiendo la cantidad de iones de hidrogeno (H+) disueltos en el agua, devolviendo valores entre 0 y 14, correspondiente 0 a un agua muy ácida y 14 a una muy básica.

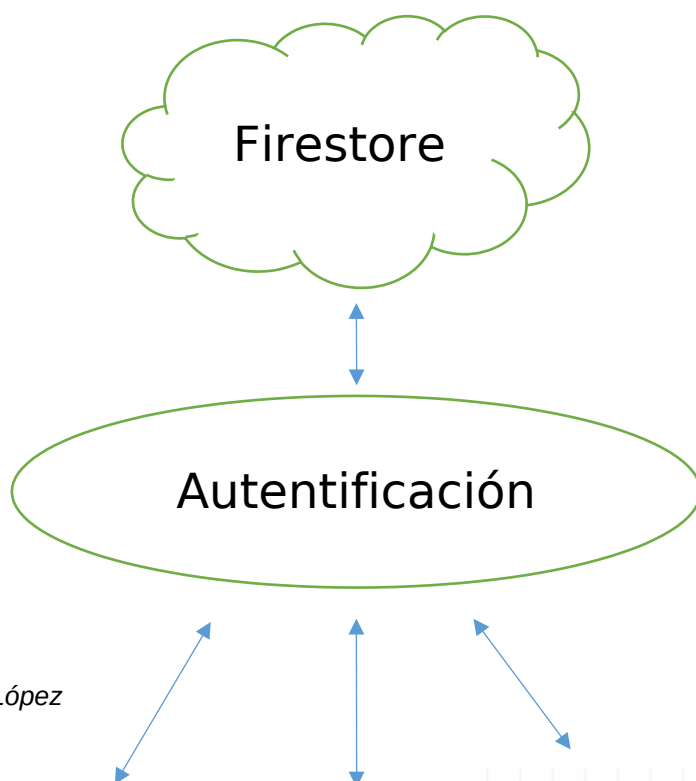
3.6.5. Sensor temperatura agua:



- ✓ Se realizan cada 5 segundos 500 lecturas y, se calculará la media de todas las lecturas, con el objeto de compensar el ruido. Esta lectura se convertirá a string y se enviará a *Firestore* por el puerto serie.

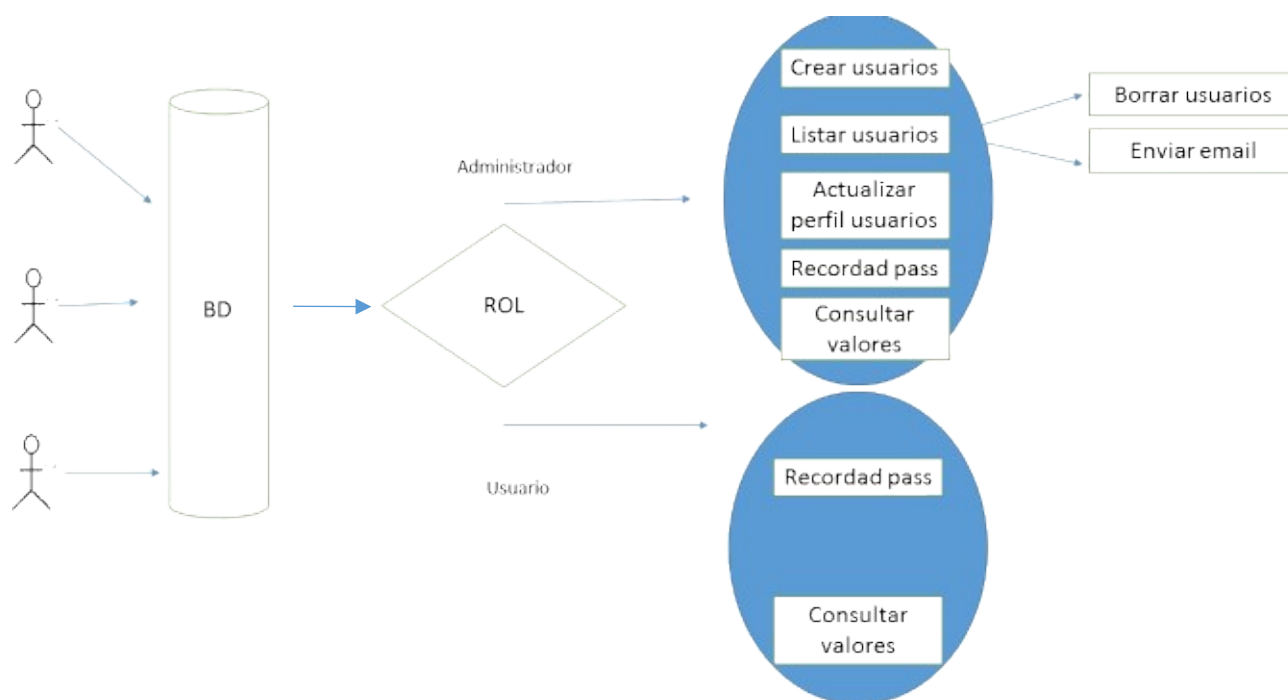
### 3.7. Casos de uso

#### 3.7.1. Autenticación:



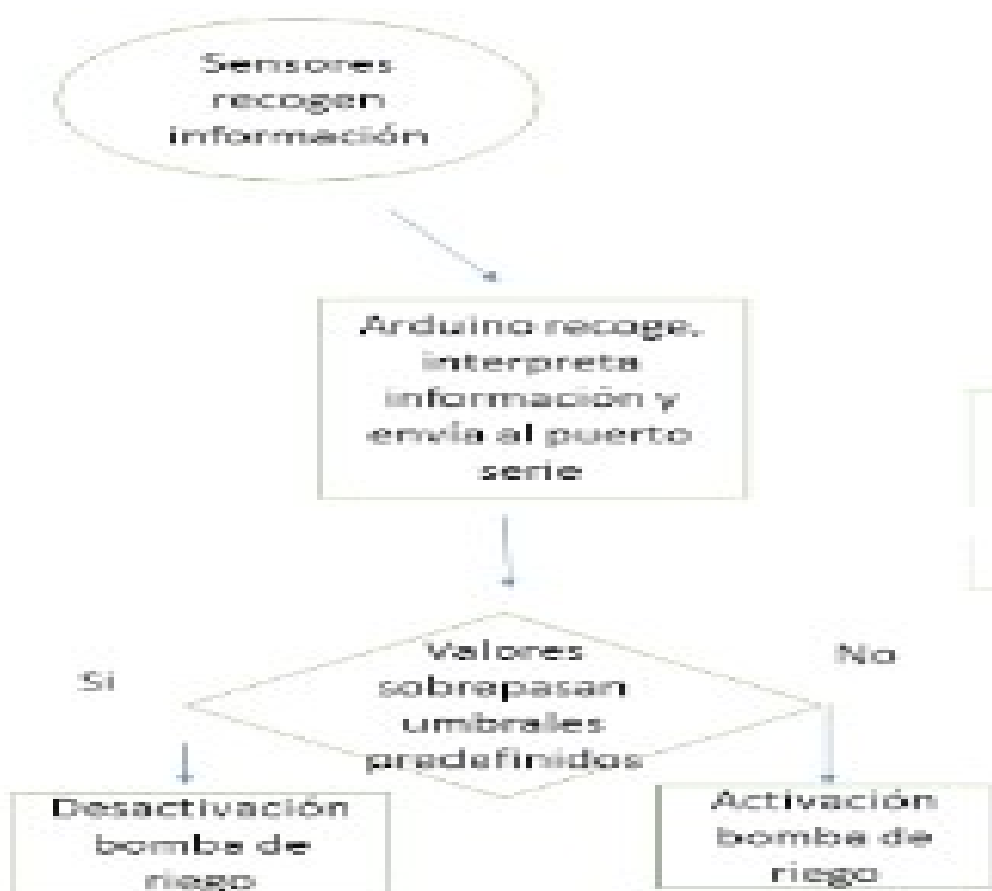
En un estadio inicial en ambos clientes, se requerirá que el usuario introduzca sus credenciales y una vez se pulse sobre el botón aceptar, se enviara una petición POST a *Firestore* y si estos datos corresponden a un usuario del sistema, *Firestore* posibilitara la lectura de datos al usuario.

### 3.7.2. Gestión de usuarios (Solo cliente web):



En este caso de uso se representa como se vera el dashboad de cada usuario en función del rol que tenga el usuario logueado.

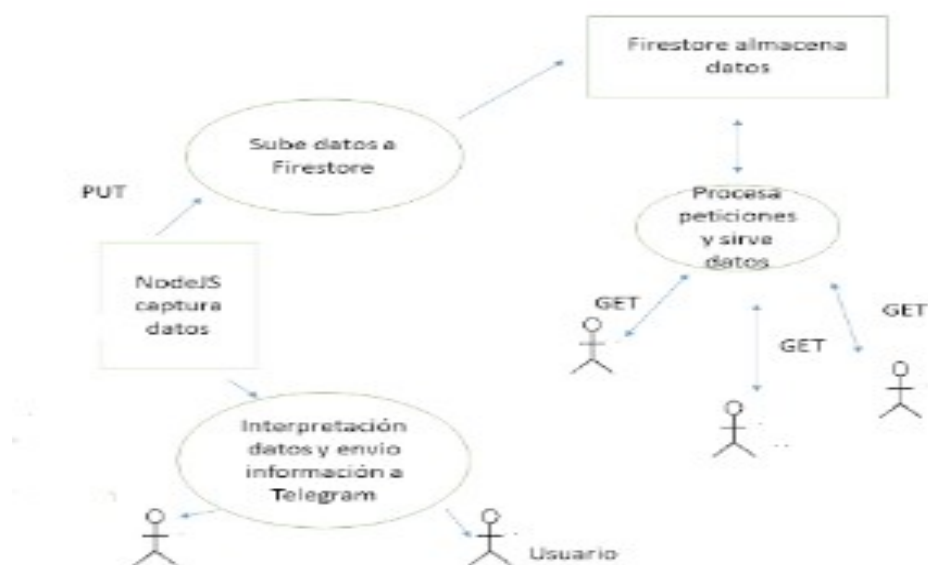
**3.7.3. Recoger valores:**



En este primer estadio del ciclo de vida del proyecto se recogerán e interpretarán, por parte de *Arduino*, las señales que desde los sensores le llegan. Asimismo, una vez recibidas e interpretadas estas señales, *Arduino* las analizará y enviará señales a la bomba de riego para que se detenga o arranque.

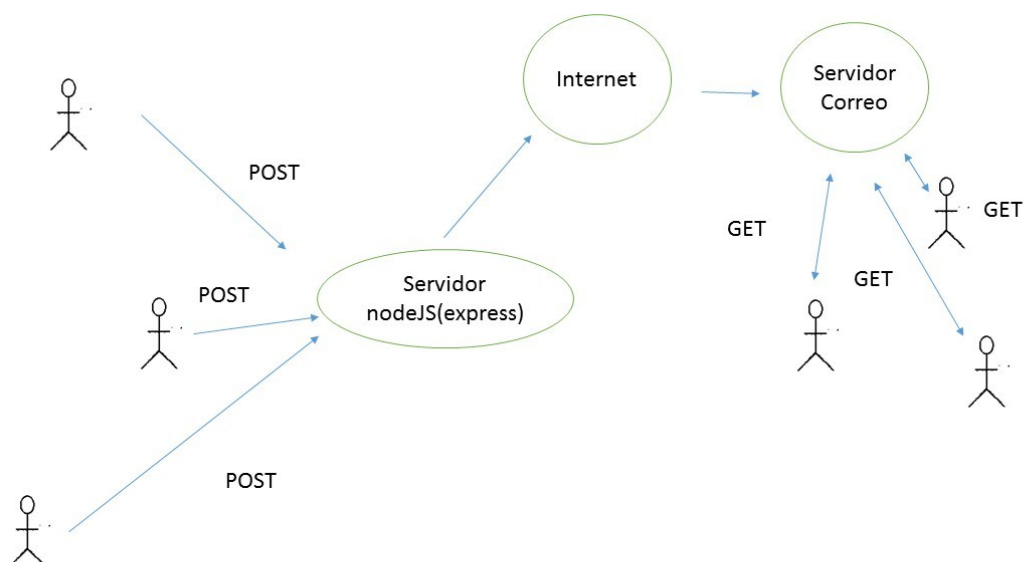


### 3.7.4. Consultar valores



En este segundo estadio, se recogerán, por parte script en *NodeJS*, tras lo cual los respaldara en BD, donde los diferentes clientes podrán consumir BD.

### 3.7.5.Caso uso enviar email (solo cliente web):



En este caso de uso se representa como es el ciclo de vida de la funcionalidad para enviar email donde, en primera instancia se le enviara a una API-REST una petición POST, con los datos del email a enviar, este a su vez se los remitira a un servidor SMTP, este a su vez enviando email a la dirección destino y en última instancia recepcionandolo por parte del usuario destino.



## 4. DESARROLLO DEL PRODUCTO SOFTWARE

### 4.1. PROGRAMACIÓN.

#### 4.1.1. Servidor de datos (Arduino)

Con el fin de agrupar todos los valores y redirigirlos al puerto serie, se ha definido una función, a la cual llamarán todas las funciones que recogen los valores que *Arduino*. Para lo cual Arduino convertirá a float el evento físico recogido por los distintos sensores. Estas funciones, llamarán a otra función que recibirá como parámetro el valor numérico recogido y lo convertirá en un string, almacenando tanto el valor numérico como el string en sendas variables globales, ya que más tarde utilizaremos los distintos valores para tomar la decisión de activar o desactivar la bomba de riego.

Una vez hecho esto se construye un único string con todos los valores recogidos y que previamente convertimos a string, enviando la cadena al puerto serie que será recogida por un script en *javascript* en la plataforma *NodeJS*.

#### 4.1.2. Respaldo de datos en firestore (Nodejs)

En primer lugar, nos descargamos el token para autenticarnos como administrador en *Firestore* y de esta forma tener permiso de escritura en el proyecto. Para lo cual, se le indica cual será el recurso de entrada, puerto serie, tras lo cual recogemos el string separado por comas, se separarán y almacenarán en variables nombradas adecuadamente, a fin de evitar confusiones, para finalmente, subirlas a *Firestore* casteadas a float. Cabe destacar que a fin de cumplir con diversas tareas se alimentarán otras dos colecciones:

1. *Values\_history*: colección que contendrá el valor acumulado de las diferentes métricas. Esto se consigue realizando un fetch sobre esta colección y sumándole el valor nuevo recibido.
2. *Values\_log*: colección que contendrá distintas lecturas recogidas a lo largo del día y que además de los distintos valores se guardarán junto con una marca temporal, con el fin de más tarde utilizar esos datos a modo de auditoría. Esto se consigue:

- a. Cuando se procede a la lectura de datos se comprueba la marca temporal de la lectura y si esta se encuentra en las marcas temporales establecidas previamente, se subirán a esta colección.

Finalmente destacar que estos valores se analizarán con el fin de crear un proceso que envía un mensaje del estado en que se encuentra la bomba de riego a un bot de *Telegram*.

#### 4.1.3. Reglas Firestone

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /users/{user} {
      allow read: if request.auth.uid != null;
    }
    match /databases/{database}/documents {
      match /values/{value} {
        allow read, write, delete, update: if request.auth.uid != null;
      }
    }
    match /databases/{database}/documents {
      match /values_history/{value} {
        allow read: if request.auth.uid != null;
      }
    }
    match /databases/{database}/documents {
      match /values_log/{value} {
        allow read: if request.auth.uid != null;
      }
    }
  }
}
```

Como se puede apreciar, en las colecciones values, values\_history y values\_log, tan solo podrán leer los usuarios logueados en el sistema, teniendo tan solo permisos de escritura el script en JS, ya que se le indico el token de escritura perteneciente al proyecto.

En la colección user los usuarios logueados tienen todos los permisos, ya que los usuarios sin rol admin, podrán:

- A. Cambiar su foto de perfil.
- B. Cambiar su password.

Y los usuarios con rol admin, además de las funcionalidades anteriores, podrán también:

- A. Consultar lista de los usuarios del sistema, donde se podrán enviar un email a todos los usuarios o bien a usuarios individualmente.
- B. Cambiar perfil usuario.

Para el sistema de login y el consumo datos por parte del cliente, se ha determinado describirlos en apartados diferentes, ya que, si bien su funcionalidad es común a sendos clientes, Android y web, y los clientes finales no notaran diferencia alguna, la forma en que recogen la información es distinta.

#### 4.1.4. Características propias del cliente **Android**:

En el caso de este cliente, la funcionalidad, si bien solo podremos acceder a resto de funcionalidades si nos logueamos con un usuario del sistema, independientemente al perfil de usuario logueado, todos los usuarios tendrán acceso a las mismas funcionalidades.

Para lo cual en todos los accesos a BD lee la colección y, para que permanezca a la escucha de BD, esperando que se produzca un cambio, se le añade el evento **addSnapshotListener**.

```
public void acquireValues() {
    final DocumentReference docRef = db.collection( collectionPath: "values").document( documentPath: "YC0HJwj1qynXC12LVpAV");
    docRef.addSnapshotListener((snapshot, e) → {
        if (e != null) {
            Log.d( tag: "agruino", msg: "Listen failed.", e);
            return;
        }

        if (snapshot != null && snapshot.exists()) {
            Log.d( tag: "agruino", msg: "Current data: " + snapshot.getData());
            value.setValue(Float.parseFloat(Objects.toString(snapshot.get(value.getKey().toLowerCase()))));
            if(!Tools.isNumeric(Float.toString(value.getValue())))
                value.setValue(0);
            updateRV();
        } else {
            Log.d( tag: "agruino", msg: "Current data: null");
        }
    });
}
```

Estas lecturas se devolverán en un **snapshot** donde nos valdremos a modo de memoria temporal, de la clase value, la cual solo contendrá dos propiedades: value del tipo double y measure del tipo string, una vez se ha cargado la clase con el valor y unidad de medida, esto se hará el en **Oncreate** del fragmento, se actualizará el layout con los valores de la clase en primer lugar.

Para la consulta de datos de la colección values\_log y, aquí quiero detenerme un poquito más en el detalle, ya que si bien no es difícil, si se ha de comprender bien todos los pasos dados.

- En primera instancia se recuperan todos los datos con el mismo método antes descrito, los datos recuperados en este caso se almacenarán en una Colección,

sirviéndonos a forma de memoria intermedia entre BD y colección, de la clase `values_log`. Una vez recuperados todos los valores de BD y almacenados en la colección, se procede a persistir está en un **VIEWMODEL**, más adelante esto cobrará sentido este paso.

```
public void acquireValues(String document) {
    db.collection(document)
        .get()
        .addOnCompleteListener(new OnCompleteListener<QuerySnapshot>() {
            String key = valueHistory.getKey().toLowerCase();

            @Override
            public void onComplete(@NonNull Task<QuerySnapshot> task) {
                if (task.isSuccessful()) {
                    for (QueryDocumentSnapshot document : task.getResult()) {
                        valueLog = new ValueLog(valueHistory);
                        valueLog.setValue(Float.parseFloat(Objects.toString(document.get(key))));
                        valueLog.setDate(Objects.toString(document.get("dateString")));
                        valueLog.setTime(Objects.toString(document.get("time")));
                        valueLog.setDateUnix(Long.parseLong(Objects.toString(document.get("date"))));
                        Log.d( tag: "agruino", msg: "aki" + valueLog.getValueHistory());
                        values.add(valueLog);
                        viewModel.setValueLog(valueLog);
                        updateRV();
                    }
                    viewModel.setValues(values);
                } else {
                    Log.d( tag: "agruino", msg: "Error getting documents: ", task.getException());
                }
            }
        });
}
```

- Estos datos recuperados, pueden ser muchos, perdiendo total o parcialmente su capacidad analítica, por lo se ha implementado un método para que estos datos se puedan filtrar por su fecha de lectura, para lo cual pincharemos sobre un icono que, está en la parte superior derecha, en donde elegiremos la fecha por la que deseamos filtrar. En este momento, recuperamos la fecha y cargamos una clase `filter`, la cual persistimos en el **VIEMODEL** anteriormente comentado, pero, en este caso cargaremos un **MUTABLELIVEDATA** del tipo `filter`. Una vez se ha cargado y persistido el filtro, dentro del **OnCreateView** del fragmento se lanzará un observe ejecutando entre otros el siguiente método:

```

//observe
viewModel.getmFilter().observe(getActivity(), (Observer) (filter) -> {

    acquireValues(DOCUMENTVALUES, PATHVALUES);
    acquireValues(DOCUMENTHISTORY, PATHHISTORY);

    mAdapter.setValues(filter(string2long(viewModel.getDate())));
    mAdapter.notifyDataSetChanged();
    rvLog.setHasFixedSize(true);
    rvLog.setLayoutManager(new LinearLayoutManager(requireActivity()));
    rvLog.addItemDecoration(new DividerItemDecoration(requireActivity(), DividerItemDecoration.VERTICAL));
    rvLog.setAdapter(mAdapter);

});
return v;
}

private List<ValueLog> filter(long date) {
    valuesFilter.clear();
    valueLog = viewModel.getValueLog();
    for (ValueLog v : values) {
        if (v.getDateUnix() >= date)
            valuesFilter.add(v);
    }
    return valuesFilter;
}
}

```

Donde, en el método **string2long**, se recupera la fecha anteriormente persistida y se transformara a formato unix, que se le pasara como argumento al método filter que nos retornara una lista con los valores filtrados por fecha.

Para hacer esto: se recupera la colección, anteriormente persistida, con todos los resultados, se compara su campo fecha con la fecha que el método **string2long** nos devolvió y si esta es mayor o igual, se añade a una colección destinada a albergar los resultados filtrados para, finalmente pasársela como argumento al adaptador para que los presente en nuestro layout.

#### 4.1.5. Características propias de cliente Web (Angular 8):

El ciclo de vida de esta aplicación comenzara enviando una petición POST a *Firestore* con las credenciales introducidas en el formulario de entrada y si estas son correctas, se cargará en memoria una clase user, para que cuando el usuario se haga unlogin esta memoria intermedia se pierda y se persista en BD. En caso de no producirse correctamente el login, se ha añadido una funcionalidad llamada **routing**, en la que por medio de los archivos de definición **auth-guard.ts** y **routes.ts**, los utilizaremos para que los usuarios no logeados no puedan acceder a la aplicación.

En caso de que el logueo sea correcto, por medio de archivos de definición: **secure-inner-pages.guard.ts** y **routes.ts**, los utilizaremos para según el perfil del usuario logeado tenga acceso a unas zonas de la aplicación.

Para consultar BD angular 8, más bien yo, he utilizado una colección de tipo observable que, realizara peticiones asíncronas al servidor, de forma que cuando este no se reciben datos de BD, la aplicación no se bloquea. El observable permanecerá a la escucha de que



se produzca un cambio en BD para actualizar su valor. Ahora se procede a consumir este servicio en un componente para lo cual debe de tener la marca **export**. Ya en el componente que va a consumir el servicio, lo importamos, se inyecta en el constructor y se implementa la interfaz **Onit**, donde igualaremos el observable a un array de tipo values, donde para que el array se cargue con los valores del observable, deberemos suscribirnos al observable.

```
export class ServiceReadValuesService {
  //array where we stored new values read from arduino
  values: Observable<ValuesInterface[]>;
  valuesDoc: AngularFirestoreDocument<ValuesInterface>;
  valuesHistory: Observable<ValuesHistory[]>;
  valuesHistoryDoc: AngularFirestoreDocument<ValuesHistory>;
  valuesLog: Observable<ValueLog[]>;
  valuesLogDoc: AngularFirestoreDocument<ValueLog>;

  constructor(public afs: AngularFirestore) {
    this.values = afs.collection("values").valueChanges();
    console.log(this.values);

    this.valuesHistory = afs.collection("values_history").valueChanges();
    console.log(this.valuesHistory);

    this.valuesLog = afs.collection("values_log").valueChanges();
    console.log(this.valuesLog);
  }
  //methods
  //method for cath values
  getValues() {
    return this.values;
  }
}
```

Imagen de cómo se debe configurar este para que el observable se ali mente de BD y sea exportable.

Este array ya si lo podremos mostrar en nuestro layout haciéndole lo que en angular se conoce como *interpolation binding*.

```
<ul *ngFor="let value of values" class="collection with-header">
  <br><br><li class="list-group"><h4 class="centerTitle">{{ 'agruino.quality' | translate }}</h4>
  <div *ngIf="value?.conductivity <= 1">
    <div class="conductivity fixed">
      <li class="list-group-item"><i class="alert alert-info">{{ 'agruino.conductivity' | translate }}
      <meter class="meter meter_meterOrange" min="0.0" max="3.0" value="{{value.conductivity}}">{{value.conductivity}}</meter>
    </div>
  </div>
</li>
</ul>
```

Para la consulta de los datos para realizar auditorías la función es muy similar a como mostramos los valores actuales, también hará uso de un servicio que recuperará todos los registros y filtraremos por la fecha de entrada, cabe destacar que la fecha comparada y la del registro estarán en formato unix..

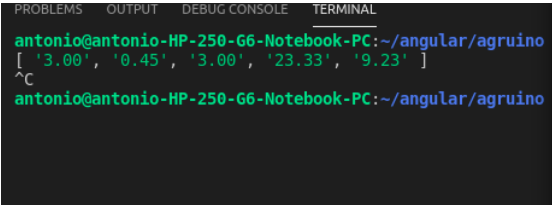
También destacable en este cliente es que se ha desarrollado un script de *NodeJS* mediante el framework *Express*, donde le haremos una petición POST con los la información que deseemos para que el cuerpo, asunto y usuarios destino de email. Este a su vez se los enviara a nuestro proveedor SMTP, enviando el email al usuario destino.

## 4.2. Pruebas:

### 4.2.1 Captura datos que *Arduino* recolecta de sus diferentes sensores a el acoplados:

```
antonio@antonio-HP-250-G6-Notebook-PC:~/angular/agruino (copia)/arduino$ node index-bis.js
[ '3.00', '0.45', '3.00', '23.33', '9.23' ]
^C
antonio@antonio-HP-250-G6-Notebook-PC:~/angular/agruino (copia)/arduino$
```

### 4.2.2. Respalidar datos en BD:



The terminal window shows the command `node index-bis.js` being executed, which outputs an array of sensor data: `['3.00', '0.45', '3.00', '23.33', '9.23']`. The user then presses `^C` to stop the process.

values	
values_histo...	conductivity: 0.45
values_log	moisture: 3
	ph: 23.33
	temp: 9.23
	turbidity: 3

### 4.2.3. Recoger datos por parte de clientes:



## 5. MANUAL DE USUARIO.

### 5.1. Aplicación web:

#### 5.1.1. Despliegue:

Previo paso a su despliegue en un servidor web, **NGIX** en este caso, se debe modificar el archivo **envirotment.prod.ts**: e introducirle los datos de conexión a BD. Ahora ya está todo preparado para una compilación para producción:

```
export const environment = {
  production: true,
  //firebase conection parameters
  firebaseConfig: {
    apiKey: [REDACTED],
    authDomain: "agruino.firebaseio.com",
    databaseURL: "https://agruino.firebaseio.com",
    projectId: "agruino",
    storageBucket: "agruino.appspot.com",
    messagingSenderId: "376909009360"
  }
};
```

, tras lo cual procedemos a compilar el proyecto con el comando ->

**ng build – prod**, esto genera una carpeta **dist** que, cuyo contenido es lo que tenemos que llevar al **document root** del servidor web.

Si estamos en una infraestructura en la nube, tipo AWS o azure y con una distro **.DEB**, en realidad nos da lo mismo ya que procedimiento es similar en todas las distros Linux, OSX o Windows. Crearemos carpeta **agruino-SRV**, carpeta que tendremos que tomar bajo la propiedad de nuestro usuario y darle todos los permisos a nuestro usuario y de lectura y ejecución a usuarios de nuestro grupo y al resto de usuarios -> que en el caso de NGIX se configura en el archivo -> **chmod –R \$USER:\$USER ~/agruino-SRV | chomd –R 755 ~/agruino-SRV**. Ahora renombramos el archivo de configuración por defecto de la carpeta **sites-aviable** a **agruino.com** -> **sudo mv /etc/nginx/sites-available/default /etc/nginx/sites-available/agruino.com**, editamos este archivo -> **sudo nano /etc/nginx/sites-available/agruino.com** y cambiamos dos líneas:

```
#root /var/www/html;
root /home/antonio/agruino-SRV;

# Add index.php to the list if you are using PHP
index index.html index.htm index.nginx-debian.html;

server_name _;

location / {
    # First attempt to serve request as file, then
    # as directory, then fall back to displaying a 404.
    #try_files $uri $uri/ =404;
    try_files $uri$args $uri$args/ index.html;
}
```

La línea root existente, la comentamos y se crea una nueva con la nueva ruta de nuestro servidor.

La segunda como se puede apreciar, el comportamiento de este, y todos, los servidores web es redireccionar a la página 404, en este caso angular se comporta de otra forma ya que, es el con el archivo **routes.tc** el que maneja las rutas, por lo que nos limitaremos a indicarle al servidor que redirija a la página index.html pasándole como argumento la URL donde queremos ir. Ahora creamos un enlace simbólico en sites-enabled del archivo agruino.com de sites-available -> **sudo ln -s /etc/nginx/sites-available/example.com /etc/nginx/sites-enabled/** y reiniciamos nginx -> **sudo systemctl restart nginx**. En este momento ya estamos listos para modificar el archivo /etc/host, para que cuando el servidor web reciba una petición a agruino.com o www.agruino.com, lo redirija hacia la dirección donde se encuentra tu servidor web, a, siempre es más fácil recordar una URL que una IP:

```
127.0.0.1      agruino.com www.agruino.com
```

### 5.1.2. Funcionalidad:

La pantalla de inicio es una pantalla de login, donde si nos logeamos correctamente, el sistema no nos permitirá acceder a ninguna parte del sistema y si nos logeamos bien, dependiendo del rol del usuario logeado, este podrá acceder a unas zonas u otras:

Panel de Control

Nuevo usuario

Usuarios

Actualizar perfil

Recordar password

Valores agua

Datos Históricos

Hola: **antocarmona@gmail.com**

User ID: **JqBy7lUGXuP**  
Rol: **Administrador**  
Email confirmado **true**

Control Panel

Forgot Password?

Water values

Historical Values

Selecciona archivo

Examinar...

No se ha seleccionado ningún archivo.

Hi: **3820619@alu.murciaeduca.es**

User ID: **Q4nZu1EanTWq8mrM1ue82LMW1**  
Rol: **User**  
Email verified **true**

Select file

Examinar...

No se ha seleccionado ningún archivo.

Update photo

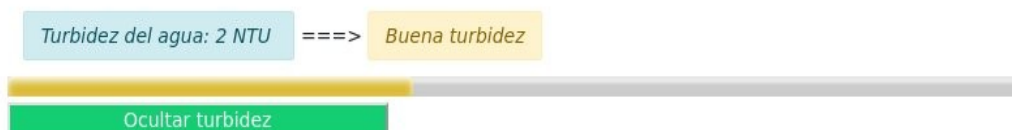
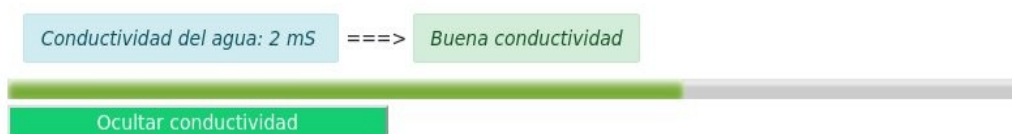
Antonio Carmona López

Pág. 38

Con ambos perfiles el sistema nos permitirá consultar el valor actual de los valores recogidos por *Arduino*:



## Calidad del agua



Mostrar ph

También, con todos los perfiles el sistema nos permitirá consultar datos históricos y estadísticos:

### AVG values

Moisture	Turbidity	Conductivity	PH	Temperature
0.02 %	0.01 MTU	0.00 MS	0.10	0.10 MTU

Date:

01 / 01 / 1970

### Historical moisture values

Value	Date	Time
5 %	01/01/2013	8:00
Value	Date	Time
5 %	01/01/2013	12:00

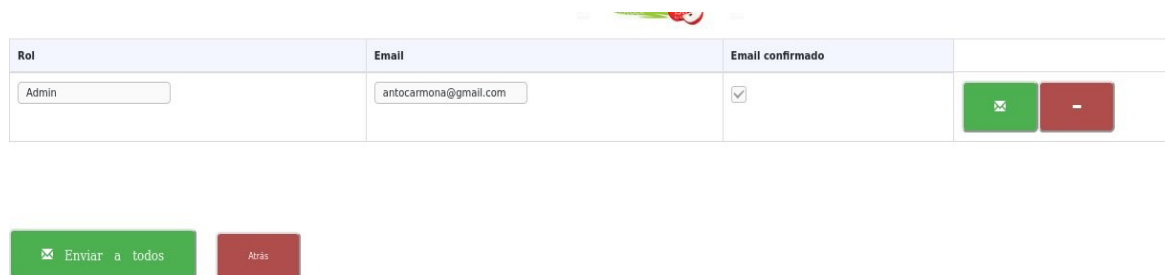
### Historical conductivity values

Value	Date	Time
0.25 MS	01/01/2013	8:00
Value	Date	Time
0.25 MS	01/01/2013	12:00

### Current moisture value

Value
50 %

En cambio, solo los usuarios con rol de administrador, podrán consultar el listado de usuarios del sistema, donde se podrá hacer distintas operaciones:



Rol	Email	Email confirmado	
Admin	antocarmona@gmail.com	<input checked="" type="checkbox"/>	<div>+</div> <div>-</div>

✉ Enviar a todos

Abrir

## 5.2. Cliente Android:

### 5.2.1. Generar paquete distribución:

[Documentación oficial.](#)

### 5.2.2. Funcionalidad:

Con respecto a la funcionalidad es muy similar al cliente web, ya que por motivos de limitación de espacio y para que la experiencia de usuario sea buena, se ha decidido que los distintos roles tengan las mismas funcionalidades, es decir: tan solo podrán consultar valores actuales, valores históricos y valores estadísticos.

Al igual que el cliente web también es multi-idioma (inglés y español).

Nada más iniciar la aplicación se nos presenta una página de login, donde si nos logueamos con un usuario del sistema este nos redirigirá a otra pantalla donde podremos ver tres iconos, uno para consultar el valor correspondiente a la humedad relativa, otro donde poder consultar los valores del agua y otro icono donde consulta valores históricos y estadísticos.

Si pinchamos sobre el segundo icono, que corresponde a los valores del agua, el sistema no redirigirá a otra pantalla, pantalla en la que poder escoger entre los distintos valores registrados.

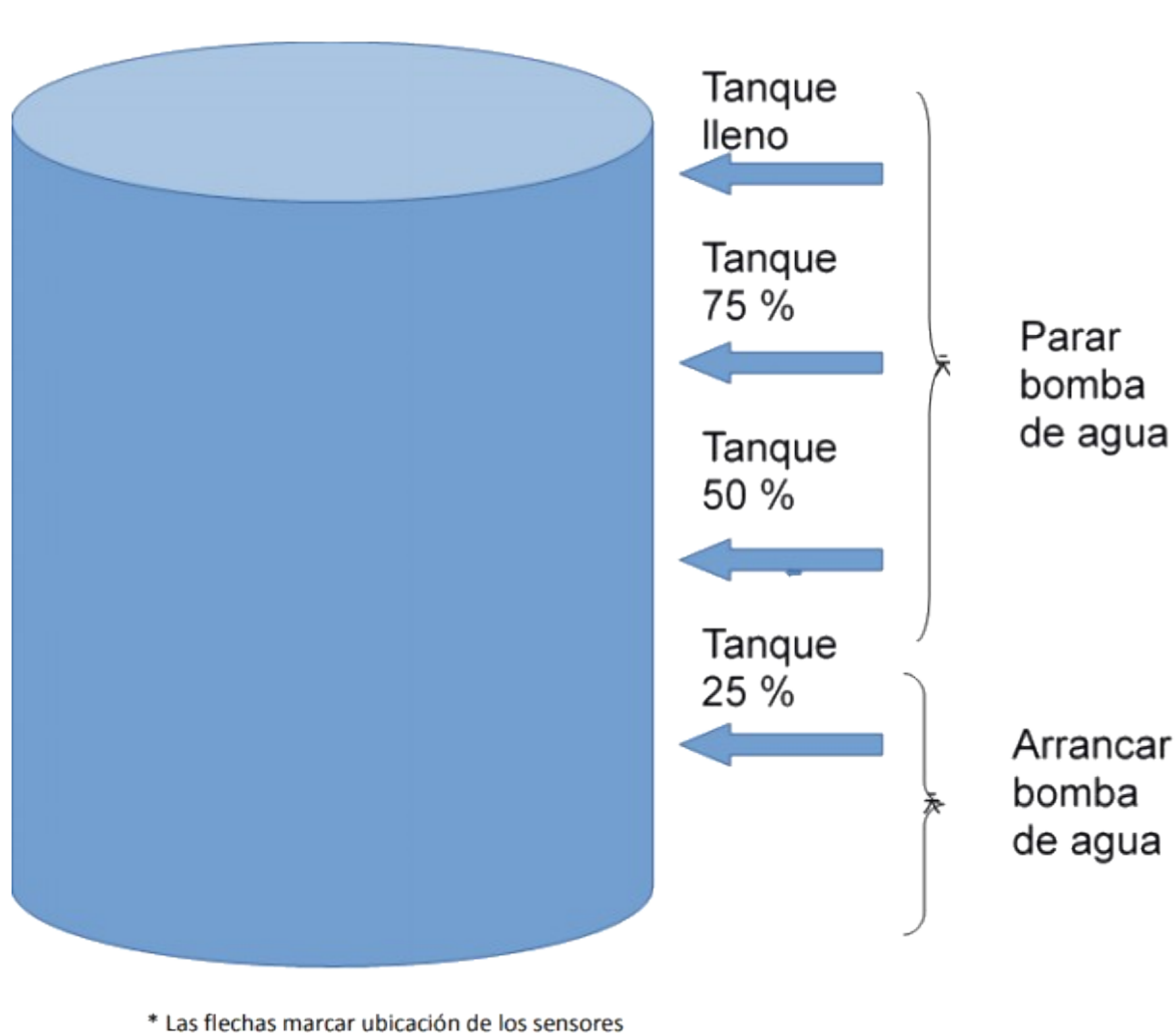
Si pinchamos sobre el tercer icono, se no abrirá otra pantalla donde escoger entre cada uno de los valores del cultivo y, si pinchaos sobre un valor cualquiera, el sistema nos redigira a otra pantalla donde podremos ver el valor actual que el sensor recogerá en el cultivo, el valor medio y lecturas históricas. En esta misma pantalla podremos pinchar sobre un icono de búsqueda, donde podremos filtrar los datos históricos por fecha y hora.





## 6. CONCLUSIONES Y VÍAS FUTURAS.

*Agruino* no pretende ser un proyecto cerrado, ya que pretende ser un proyecto vivo y no solamente recoger una foto fija y, por esta razón, este proyecto se ha intentado desarrollar de forma totalmente escalable manteniendo la estabilidad total, teniendo casi exclusivamente como factor limitante el servidor donde estará alojada la BD. Teniendo en cuenta todos estos factores, *Agruino* se ha desarrollado de forma que el proyecto sea susceptible a cualquier modificación/mejora de forma poco traumática de cara al desarrollador y usuario, por ejemplo, una posible mejora sería la de motorizar el nivel de la balsa de riego, esto se podría hacer con un coste casi nulo. Una forma sencilla de hacer esto, sería la de [construir](#) tantos sensores como se deseen colocar en la balsa de riego, estos sensores son iguales al sensor de conductividad, siendo incluso más fácil su montaje, ya que en este caso no haría falta utilizar resistencia alguna, ya que no se precisan de lecturas precisas, tan solo se desea saber si existe conducción entre electrodos, lo que significaría que está sumergida en agua, o no existe conducción. Con esta información se pueden tomar decisiones, como la activar bomba de agua para llenar tanque o parar bomba una vez que esté lleno.



Otras mejoras, si bien fáciles de implementar, si se requerirá cambiar hacia una placa *Arduino* con más pines (*Arduino mega*, por ejemplo) u otras placas *Arduino* (*Arduino uno*) en paralelo, podría ser: La utilización de los umbrales que se han definido en las diferentes lecturas, con el fin de tomar decisiones dirigidas en un sentido u otro en función del umbral rebasado. Pero esta funcionalidad tal y como se ha comentado, si bien no sería difícil de implementar, sí que requeriría de más medios técnicos/económicos, que quizá fueran en contra de la filosofía del proyecto. A su favor queda, que la implementación de esta mejora, sería dar un paso más hacia la automatización casi total de una explotación agrícola, facilitando de esta forma el trabajo del responsable

- Este sensor se ha fabricado de la siguiente forma:
  - ✓ Se coge un clip para sujetar folios de papel, se estira y se corta por la mitad.
  - ✓ Cada uno de los trozos se unen a los extremos de un cable. o Ambos “electrodos” se fijan al esqueleto de un bolígrafo.
  - ✓ Para finalmente conectar los otros extremos del cable a *Arduino*.

[VOLVER sensor conductividad](#)

[VOLVER sensor riego](#)



## 7. REFERENCIAS Y BIBLIOGRAFÍA.

[Documentación oficial \*Arduino\*.](#)

[Documentación \*Android\*](#)

[Documentación \*Firestore\*.](#)

[Crear boot \*Telegram\*.](#)

[Documentación de \*Bootstrap\*](#)

[Documentación oficial \*Angular\*](#)

[Video tutorial CRUD \*Angular 8\* DOMINI CODE](#)