ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
Ingeniería del Software

Aplicación de sistema de control y mantenimiento del nivel de CO2 en espacios cerrados.

System of control and maintenance of the CO2 level in encolsed spaces.

Realizado por
Sergio Ríos López
Tutorizado por
Luis Manuel Llopis Torres
Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, Septiembre de 2018

Fecha defensa:
El Secretario del Tribunal

Resumen: El siguiente proyecto consiste en el diseño, implementación y prueba de un sistema automatizado del control de la calidad del aire en espacios cerrados usando componentes de bajo coste.

La realización del proyecto se ha dividido en cuatro fases diferentes; una primera fase de elección de las tecnologías a utilizar basándose en una idea general del proyecto, una de diseño técnico de las partes más importantes del proyecto, el propio desarrollo del proyecto incluyendo tanto el desarrollo software como del circuito electrónico, y por último una fase de pruebas donde se ha comprobado que todo funcionara como se esperaba.

El proyecto está dividido en cuatro módulos trabajando entre tres máquinas diferentes, cada uno encargado de una parte fundamental en la ejecución del conjunto. Estos son un módulo de control del nivel basado en Arduino, encargado de mantener los niveles de CO2 por debajo del valor deseado mediante la activación y desactivación de un actuador o ventilador, una aplicación web ejecutada en un PC que servirá como vía de comunicación con el usuario de un modo bidireccional, de modo que este pueda tanto visualizar datos como alterar al opciones del sistema, un servicio web ejecutado en una Raspberry Pi que mandará ordenes al módulo de control actuando como intermediario entre Arduino y la aplicación web, y una base de datos encargada de mantener el modelo de la aplicación web y guardar todos los datos referentes al funcionamiento del sistema.
Todo esto estará aunado en el proyecto final en una pequeña maqueta que simulará una habitación cerrada con en la que se insuflará aire, observando así el correcto funcionamiento del sistema en su conjunto.

Palabras clave: JSF,Restful,Raspberry, Arduino,CO2,Serie,UART,Web,Servicio

Abstract: This project consists in the design, implementation and tests of a system for the control of the air quality in enclosed spaces using low cost components.

This project's realization has been divided into four different phases; a first technologies-to-use election phase based on a general idea of the project, one of technical design of the more important parts of the project, the development itself including the software development and the electronic circuit, and finally a test phase where it has been checked that everything works has expected.

The project is divided in four modules working on three different machines, each one responsible for one fundamental part in the execution. These modules are the level control based on Arduino, in charge to maintaining the CO2 levels beneath the desired value by the activation and deactivation of an actuator or fan, a web application executed on a PC that will serve as an interface for the use in a bidirectional way, meaning that you can both visualize and change some system options, a web service running on a Raspberry Pi that will send orders to the control module acting as and intermediary between Arduino and the web application, and a data base in charge to maintaining the web application model and to save all the data concerning to the operation of the system.

All of this will be brought together into the final project as a little model that will simulate an enclosed room where it will be breathed some air, watching the correct working of the whole system.

Key words: JSF,Restful,Raspberry, Arduino,CO2,Serial,UART,Web,Service

Índice

1 Introducción

El aire, fuente de uno de los más importantes elementos para la vida, es cada vez más pobre por culpa de los diferentes agentes contaminantes cada vez más presentes en nuestras vidas[1]. Vehículos motorizados, grandes granjas ganaderas[2], fábricas,... Todo esto hace que la calidad que respiramos disminuya con el paso del tiempo y nos propicie, cada vez más, a una pérdida en la calidad de vida[3]. Esto es mucho más obvio en zonas pobladas, donde es mucho más frecuente encontrar cada uno de estos agentes contaminantes y se ve agravado, por ejemplo, en ciudades masificadas. Grandes ciudades donde el número de personas y sus actividades aumentan este impacto, provocando niveles de contaminación especialmente preocupantes.

Pero, ¿y si además nos encontramos en sitios cerrados? Sitios con poca o ninguna ventilación donde la calidad del aire, ese aire que ya de por sí flaquea, disminuye con el paso del tiempo a causa de las concentraciones de CO2, y donde no siempre es posible mantener un flujo de constante con el exterior[4]. Muchas veces encontramos carísimos y aparatosos sistemas de ventilación o aire acondicionado a los que no todos tienen acceso ya sea por la propia instalación o por los altos costes de energía que conllevan. Además, estos sistemas no están preparados más que para controlar la temperatura ambiental, y en ningún momento asegura una calidad del aire óptima .

En este trabajo, se propone una solución barata, de instalación sencilla, inteligente y configurable que permitirá asegurar espacios bien ventilados alterando lo mínimo posible el resto de parámetros ambientales.

1.1 Objetivos

El objetivo de este proyecto será el del diseño, desarrollo y prueba de funcionamiento del sistema propuesto, asegurar un costo asequible para el monto total del proyecto y mantener la independencia entre módulos para facilitar futuras mejoras en el desarrollo.

La prueba consistirá en la simulación de un ambiente cerrado a pequeña escala donde se introducirá aire expirado, subiendo el nivel de CO2 y comprobando los niveles captados y la correcta actuación del sistema de control. Además se le mandarán varios comandos desde la aplicación web para demostrar el flujo de control completo, además de solicitarle datos que mostrar en las gráficas. También se demostrará el funcionamiento de la web de control

probando entre otras cosas la identificación con diferentes credenciales y las diferentes funciones de gestión del sistema.

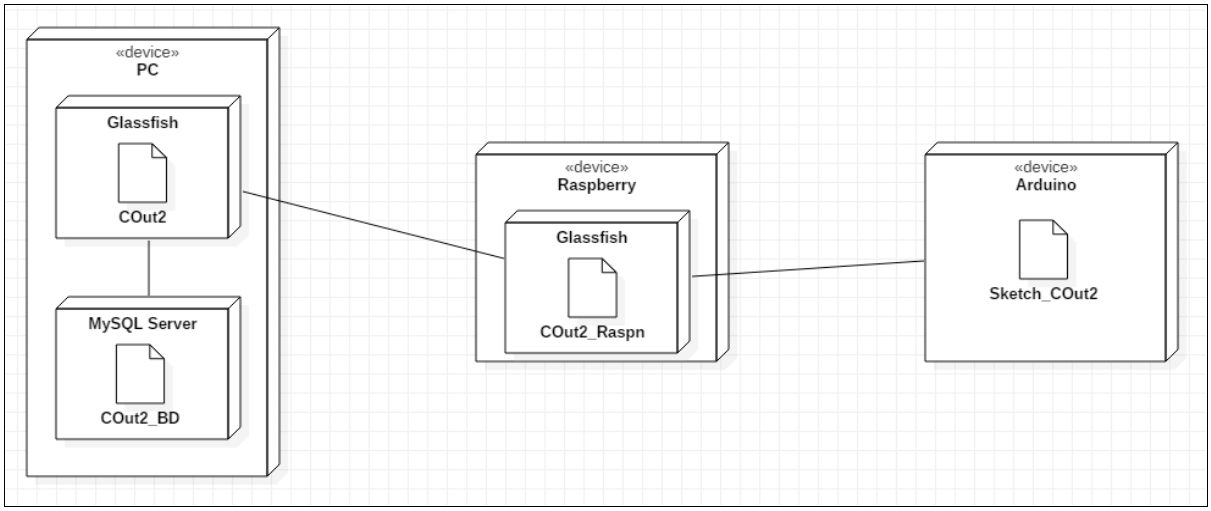

Figura 1. Diagrama de despliegue.

2 Elección de tecnologías

Debido a la necesidad de cooperación entre varios módulos diferentes, la elección de las tecnologías adecuadas para una comunicación eficaz y eficiente ha de ser una prioridad, asegurándose además de que para cada una de las partes se seleccione una tecnología coherente con la tarea que debe desarrollar.

Por ello, en el siguiente apartado se van a exponer las tecnologías distintas que forman parte de este proyecto y, si fuera el caso, otras opciones que se hayan tenido en cuenta.

2.1 Herramientas de diseño

2.1.1 Pencil Project

Pencil es una herramienta para crear '*mockups*' de interfaces de usuario de manera gratuita, fácil y rápida.

Su funcionamiento es sencillo, se dispone de un conjunto de elementos que representan los diferentes componentes de una interfaz y de una página en blanco, y el usuario dispone de la manera que mejor estime oportuna dichos elementos para sacar una representación en forma de boceto de la interfaz a realizar.

2.1.2 Fritzing

Es un programa de código abierto que permite el diseño y emulación de sistemas electrónicos basados en hardware como Arduino. Además, dispone de un editor de código de Arduino capaz de ser cargado en nuestra propia placa.

Aún así, la única función que se usará es la de diseño de circuitos, para representar de manera muy intuitiva el sistema que se pretende diseñar y ver si es posible llevarlo a cabo.

2.1.3 StarUML

Herramienta de desarrollo de modelos UML. Aunque la versión actual es de pago, existe todavía una versión de código abierto disponible.

Tiene soporte para numerosos tipos de diagramas, como pueden ser el de clases, el de flujo o el de estados. Tiene un funcionamiento sencillo y una interfaz bastante intuitiva y es una

muy buena alternativa ante otras aplicaciones de pago y está soportado por multitud de sistemas operativos.

2.2 Entornos de desarrollo

2.2.1 NetBeans

Es un entorno de desarrollo libre, de código abierto y gratuito. Actualmente pertenece a Oracle.

Entre sus numerosas ventajas, dispone de soporte para un gran número de lenguajes, entre los que se encuentra Java, HTML y CSS. Desde la versión 6.5.2, tiene soporte para el desarrollo de aplicaciones basada en Java EE, permitiendo el desarrollo y compilación de estas aplicaciones. NetBeans permite gestionar el estado de diferentes tipos de servidores de aplicaciones como Tomcat, WildFly y GlassFish, así como el despliegue y depuración de aplicaciones.

Además, dispone de un gran número de "plugins" o extensiones que permiten la integración con diferentes sistemas de control de versiones como Subversion y Git.

2.2.2 MySQL Workbench

MySQL Workbench es un editor visual de base de datos con herramientas para la construcción, edición y mantenimiento de una base de datos. Dispone de una versión *open source* totalmente gratuita y es uno de los gestores de bases de datos más utilizados en la actualidad.

Sus funcionalidades principales son:

- Herramienta de diseño del modelo de datos.
- Visor y editor de datos .
- Editor de consultas SQL.
- Monitorización del estado y rendimiento.

2.2.3 Arduino IDE

Es el IDE desarrollado y distribuido por Arduino de manera gratuita y está específicamente pensado para el desarrollo de aplicaciones para las placas del mismo nombre. El lenguaje

que usa Arduino está basado en el entorno de Processing y en la estructura Wiring, usando una versión simplificada de C++.

2.3 Módulo de control

Este módulo es será en última instancia el encargado de llevar a cabo la tarea de control que se espera del sistema. Como funcionalidades principales, tiene tanto la monitorización de los niveles de calidad como la programación de las acciones de comprobación y actuación sobre el entorno.

Por lo tanto, se ha elegido basar este módulo en una placa Arduino UNO, pensada para aplicaciones en tiempo real y que precisen de interacciones con componentes electrónicos de manera sencilla.

2.3.1 Arduino UNO

Arduino UNO es un microcontrolador basado en ATmega328P desarrollada por Arduino.

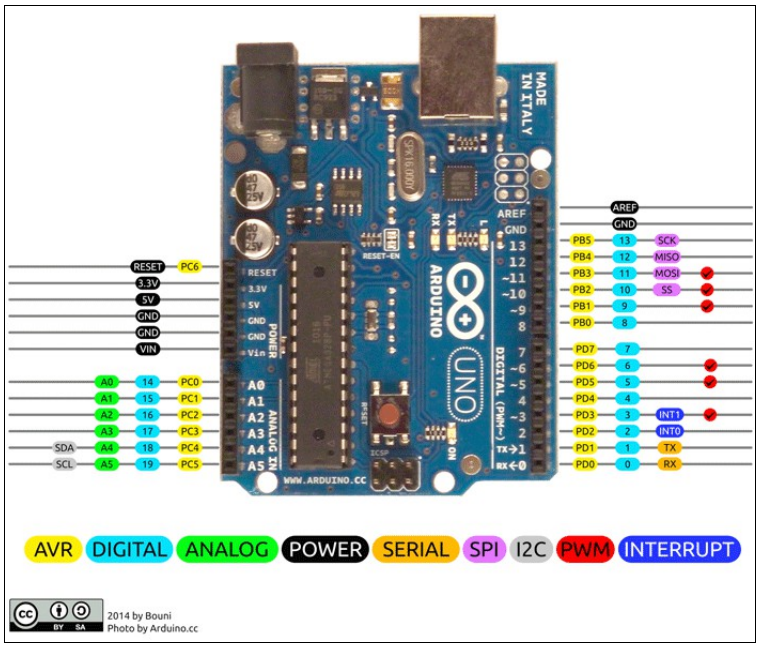

Figura 2. Disposición de los pins I/O en Arduino UNO. Bouny (2014). Recuperado de "https://components101.com"

Dispone de varios pins de entrada/salida tanto analógicas como digitales (en concreto 14 pins digitales y 6 analógicos de sólo lectura) que permiten la interacción y lectura con sistemas electrónico. Además, dos de estos pins digitales (etiquetados con Rx y Tx) permiten la comunicación serial con otros dispositivos a través de la unidad UART (*Universally Asynchronous Receiver/Transmitter*) de la que dispone. Esta comunicación es posible no sólo a través de estos pins, sino a través del puerto USB, por el que además tenemos la opción de

alimentar al sistema. También cuenta con una toma de tensión exclusiva para dicho propósito en caso de necesitar el puerto USB para otras tareas.

Además de esto, dispone de varios pins de salida de tensión de 5 V y pins GND o tierra. También cabe remarcar que se disponen de 32 kB de almacenamiento para el programa que queramos albergar, cosa que hay que tener en cuenta a la hora de optimizar el código.

2.3.2 Sensor MQ-135

Debido a la naturaleza del proyecto, esta es una de las piezas más importantes (si no la que más) a la hora de definir la propia identidad del sistema.

Se estudiaron varias posibilidades como el MG811 o el MH-Z14A, ambos sensores de CO_2 con características idóneas para la finalidad del proyecto, pero debido a alto coste, superior incluso a la suma del resto de componentes (entre 50 y 60 euros), se consideró que rompía con la idea de economizar el sistema, por lo que se se optó por la alternativa MQ-135.

Este sensor no es puramente un sensor de CO2, sino que es sensible a varios elementos que puedan alterar la calidad del aire, entre otros: NH3,NOx, Alcohol, Benceno, Humo y CO2. Si bien no se adapta completamente a la misión original buscado, puede resultar incluso mejor, ya que el lo que se pretende en última instancia es procurar un ambiente sano y libre de agentes perjudiciales, y este permite actuar aun en mayor medida en este sentido. Siguiendo los datos del *datasheet*, se puede configurar para que la estimación de la muestra sea tomada en base a ppm de CO_2, siendo así posible tomar valores precisos de este ante la ausencia de cualquier otra sustancia a la que sea sensible. Además, el coste de las distintas variantes de este sensor ronda entre 3 y 4 euros, lo que nos permite mantener un coste bajo.

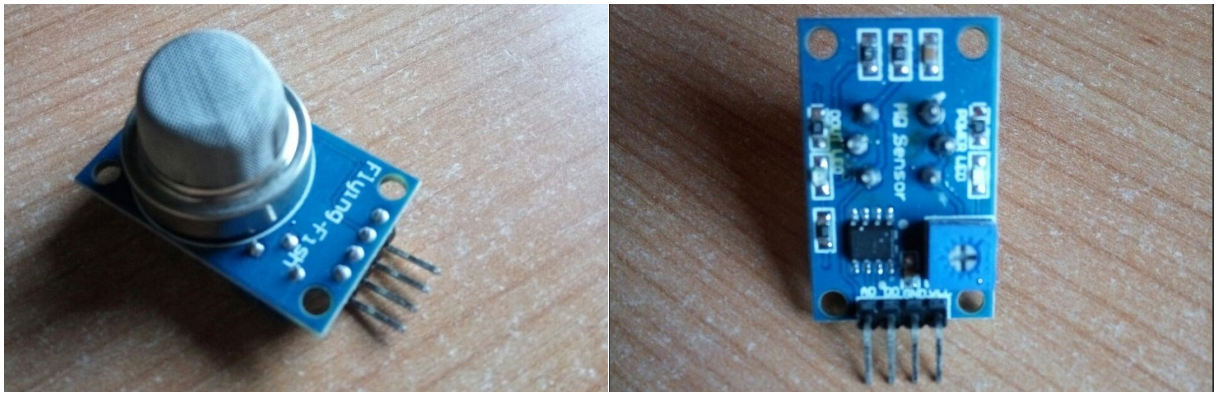

Figura 3 – Sensor MQ-135 utilizado en el proyecto.

2.4 Servicio Web

Se ha precisado de un sistema de implementación fácil, rápida y que fuera capaz de adaptarse con nuevos componentes y aplicaciones de manera sencilla. SOAP cuenta con mayor número de ventajas en el ámbito de la seguridad, pero se optó por usar RESTful porque permitía una mayor flexibilidad a la hora de implementar el cliente.

Además, como formato para la comunicación de resultados, se he tomado la decisión de usar JSON, de nuevo por su versatilidad, el gran número de herramientas gratuitas de las que se disponen para trabajar con el.

2.4.1 Raspberry Pi

Raspberry Pi es un SBC (de las siglas en inglés *Single Board Computer*) de tamaño reducido, una dispositivo que dispone de las prestaciones básicas de un ordenador concentrados en una sola placa con la intención de mantener un coste bajo.

Existen numerosos modelos de Raspberry Pi, pero se usará el modelo '*Rapsberry Pi 3 B*', uno de los últimos modelos del mercado que, pese a poseer numerosas características y un gran potencial, mantiene un coste muy reducido con respecto a sus posibilidades.

Algunas de sus características técnicas son:

- Procesador de 64 bits con arquitectura ARMv8 con dual-core
- SDRAM de 1G
- 40 GPIO (*General Purpose Input/Output*)
- 4 USB 2.0
- Ethernet, LAN inalámbrico y Bluetooth

El sistema operativo debe ser cargado en una tarjeta microsd, ya que no dispone de unidad de almacenamiento, y por lo tanto la capacidad disponible estará limitada por dicha tarjeta. Se optará por usar unas de las distribuciones de Debian adaptadas exclusivamente para Raspberry Pi conocida como "Raspbian", disponible gratuitamente para la descarga desde la propia web de Raspberry.

Dos de los GPIO son los etiquetados como Rx y TX, y serán los que se usarán para realizar la conexión con Arduino mediante el dispositivo UART del que dispone.

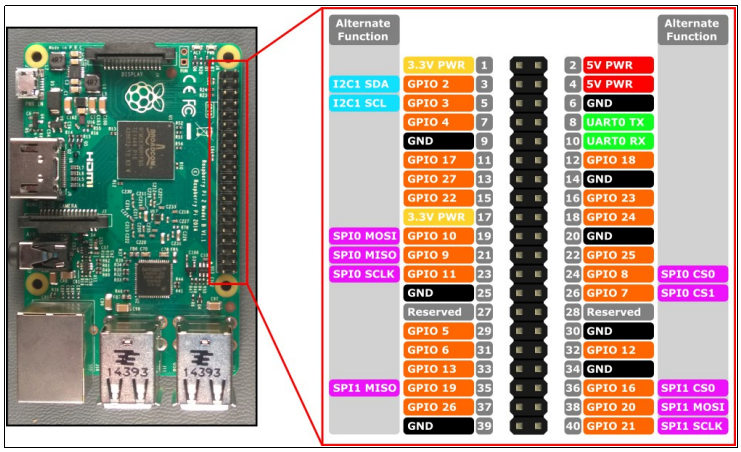

Figura 4. Raspberry Pi 3 B y mapeado de su GPIO. Recuperado de "https://docs.microsoft.com"

2.4.2 Java

Java es un lenguaje de propósito general y actualmente desarrollado por Oracle. Es un lenguaje orientado a objetos y es ejecutado por una máquina virtual, permitiendo que una misma compilación pueda ser ejecutada en una numerosa cantidad de dispositivos.

Java estructura su código en torno a diferentes clases. Estas clases están compuestas por propiedades, que mantienen el estado del objeto (así es como se llama la instanciación de una clase), y funciones o procesos, que ejecutan un serie de comandos que pueden o no alterar el estado del objeto y devolver valores tras su ejecución. Las propiedades de una clase están definidas por un tipo que puede ser otra clase, de manera que las clases instancian objetos de otras clases para así crear aplicaciones más complejas con implicaciones mayores.

Java es uno de los lenguajes más utilizados y, en consecuencia, su comunidad es una de las mayores en el ámbito del desarrollo de software. Esto es en parte debido a la facilidad de su aprendizaje, la posibilidad de ser multiplataforma y el gran número de ámbitos a los que es posible aplicarlo.

2.4.3 Glassfish

Glassfish es un servidor de aplicaciones gratuito y código abierto. Su autoría pertenece a Oracle, al igual Java, y está desarrollado para ejecutar aplicaciones que siguen las especificaciones de Java EE.

Admite el uso de unidades de persistencia como TopLink o EclipseLink, lo que permite la conexión conexión con una base de datos mediante un pool de conexiones.

2.4.4 Servicio RESTful

Los servicios RESTful son todos aquellos servicios que siguen el estilo de arquitectura REST, basada la idea de un protocolo sin estado, interoperabilidad entre servicios y universalidad de sintaxis. Este sistema basa sus llamadas en peticiones HTTP en las que la URI identifica el recurso al que se tiene acceso, por lo que se deben seguir una serie de normas para mantener un servicio ordenado e intuitivo. Además, tanto las llamadas como las respuestas son representaciones textuales de los objetos o valores a los que se hace referencia.

REST (del inglés *Representational State Transfer*) define una serie de condiciones bajo las que estructurar los servicios de manera que se cumplan las especificaciones siguientes:

- Una interfaz uniforme que mantenga una coherencia en la manera de acceder a los servicios, tanto en la nomenclatura de las URIs como en el formato de los datos (si se va a usar JSON o XML por ejemplo).
- Separación de Cliente/Servidor manteniendo la independencia entre ambos. De esta manera será posible la sustitución de uno de ellos (o incluso de ambos) y mientras el canal de comunicación (la interfaz) se mantenga igual, y ambos seguirán entendiendo la comunicación.
- Protocolo sin estados, lo que significa que cada llamada es independiente de las demás y el servidor no tiene constancia de interacciones anteriores con el cliente (los mismo pasa de manera contraria), lo que significa que la llamada debe llevar toda la información necesaria para que la petición sea entendida, sin presuponer interacciones anteriores.
- Estructurado en capas en las que un servicio se sirve de otros para realizar la petición, realizando cada uno una tarea en concreto. De esta manera, el cliente no sabe exactamente cuantos servicios entran en juego y en qué punto de la cadena se encuentra aquel con el que ha contactado. Esto aporta una cierta seguridad, aunque puede no ser viable para sistemas pequeños.
- El servidor debe saber si la información enviada es cacheable (guardar en el cliente una cierta información que previamente se ha solicitado), en cuyo caso se le debe asociar un número identificativo a dicha versión de los datos para evitar futuras peticiones de los mismos datos por parte del cliente. En caso de que dichos datos cachados hayan expirado, el cliente debe saberlo para poder solicitarlos de nuevo.

Por lo tanto, debido a la intención de que futuras ampliaciones de este proyecto puedan consistir en implementaciones de nuevos clientes que hagan uso de estos servicios, seguir una estructura REST es una de las mejores maneras de asegurar una fácil adaptación e implementación de dichos sistemas.

2.4.5 JSON

JSON (del inglés *JavaScript Object Notation*) es un formato de intercambio de información ligero basado en la sintaxis te representación de los objetos en Javascript.
Es un formato que tiene la ventaja de ser fácilmente entendible por las personas pero a su vez es fácil de interpretar por las máquinas, lo que lo hace sencillo de usar y a su vez eficiente.

Hay solamente dos tipos de estructuras básicas en JSON:

- Los objetos, que son representado por una serie de parejas clave-valor separados por comas y entre corchetes. Cada clave-valor se representa de la siguiente manera: '*clave' : valor*. El valor puede representar un *String* (cadena de texto), en cuyo caso deberá ir entre comillas simples ('), un número, un valor booleano (*true* o *false*), otro objeto, un *Array* (colección de valores) o *null*.
- Los Array o colecciones de valores, que son representados por una serie de valores ordenados separados por comas, y todo entre corchetes. Dichos valores, a su vez, pueden representar cada uno de los tipos de valores citados anteriormente.

Con estas dos estructuras universales, JSON es capaz de representar la mayoría de las estructuras de datos y por lo tanto es fácilmente implementable por casi todos los lenguajes a la hora de estandarizar un formato de comunicación, por ello es de los más utilizados en la actualidad y soportado por múltiples lenguajes, específicamente en el entorno web.

2.5 Base de datos

De entre todos los gestores y modalidades de bases de datos se ha escogido una sql de tipo relacional ya que se requería mantener una estructura de datos integra, de operaciones rápida y además en la que se tuviera experiencia. Uno de los inconvenientes de este tipo de bases de datos es la escalabilidad, pero debido a que la magnitud del proyecto no es exigente en este aspecto, no se ha tenido en cuenta como desventaja. En concreto se ha elegido usar MySQL, una herramienta gratuita y en la que se tenía bastante experiencia.

2.5.1 MySQL

Es un sistema de de base de datos relacional de los más populares del mundo. Es de código abierto y forma parte del conglomerado de tecnologías pertenecientes a Oracle.

Se caracteriza por ser base de datos con una lectura rápida y con un tamaño reducido, pero en entornos con una gran concurrencia de peticiones de alteración de los datos puede resultar en errores de integridad. Por ello es uno de los principales lenguajes utilizados en el desarrollo de aplicaciones web, debido a que generalmente la concurrencia de modificaciones de la base de datos es baja, y no supone un riesgo para su problema con la integridad de los datos. Además es ejecutable en prácticamente todos los sistemas operativos existentes y permite la creación de diferentes perfiles con diferentes privilegios.

Aunque generalmente se le asocia a PHP, también dispone del correspondiente conector de base de datos para Java.

2.6 Aplicación Web

Para el sistema de control de cara al usuario, se decidió hacer una aplicación web basada en JSF, ya que de todas las tecnologías posibles era una de las que más experiencias se contaba.

JSF además, apoyado en librerías como Primefaces, hace posible un rápido desarrollo de interfaces funcionales y llamativas, sin la necesidad de grandes archivos de estilo CSS, siendo de nuevo una característica muy importante a la hora de plantear un proyecto como este con varios módulos y un tiempo muy limitado.

2.6.1 JSF

JSF (*JavaServer Faces*) es un framework basado en el patrón MVC (*Modelo/Vista/Controlador)* para el desarrollo aplicaciones web basadas en Java. Está pensado especialmente para simplificar el desarrollo de interfaces de usuario mediante una serie de librerías que usan páginas JSP (JavaServer Pages) para desplegar las páginas, pero sin dejar de lado la el servidor, para el cual también aporta una serie de funcionalidades muy importantes como pueden ser los Beans administrados o 'ManagedBeans'.

Las siguientes son el conjunto de características que forman parte de JSF:

- Un compuesto de componentes similares a etiquetas HTML (del inglés *HyperText Markup Language*), pero con funcionalidades añadidas tales como ser capaz de

manejar eventos, alterar su estado conforme al estado del servidor, validar en el cliente los datos introducido por el usuario,...

- Un conjunto de componentes con algunas de las funciones anteriormente descritas ya por defecto por defecto.
- Forma parte del estándar J2EE al usar JSP para generar las vistas.
- Un conjunto de etiquetas para la gestión de los llamados '*ManagedBeans'* o '*Beans* administrados'. Estos son clases del servidor que se asocian a cierto ciclo de vida (como pudiera ser el de una vista, una sesión o directamente la aplicación entre otros) y que generalmente se asocian a una página haciendo que esta sea una representación del estado bean y sus posibles funcionalidades para el usuario.
- Una serie de etapas en el procesamiento de peticiones entre las que se encuentran el procesamiento de la petición y validación de los valores enviados, la reconstrucción de la vista,...
- Es extensible con diferentes librerías tanto en el lado del servidor (como es de suponer ya que sigue siendo una clase Java) como en el lado de la vista con librerías como pueden ser IceFaces, Richfaces o, la que se usará en este proyecto, PrimeFaces, un conjunto de componentes modificados basados en los que se encuentran por defecto y algunos añadidos que mejoran o simplemente aumentan las posibilidades ofrecidas por los estándar de JSF. En la mayoría de sus componentes viene soportado AJAX por defecto, mientras que de manera general hay que añadirlo explícitamente si no usamos estas librerías.

2.6.2 AJAX

AJAX (del inglés *Asynchronous JavaScript And XML*) es una técnica que, como su propio nombre indica, se vale de JavaScript para crear aplicaciones web en las que ciertas interacciones del usuario no precisen de la carga completa de la página, sino que se seleccionan ciertas secciones a 'refrescar'. Al tomar parte JavaScript, cierta parte de la aplicación se ejecuta en el cliente mientras se hacen peticiones al servidor en segundo plano, por ello se considera una comunicación asíncrona, no siendo necesaria la espera de la resolución de la petición para seguir interactuando con la aplicación.

Las peticiones asíncronas con el servidor se llevan a cabo mediante el uso de un objeto XMLHttpRequest (*Extensible Markup Language / Hypertext Transfer Protocol*), una interfaz para las comunicaciones HTTP y HTTPS.

AJAX puede suponer una gran mejora en la eficiencia y la experiencia de usuario de una aplicación, pero puede llevar a diferentes errores si no se usa con conocimiento, ya que debido a la naturaleza de los navegadores, ciertas acciones como la 'vuelta atrás' que generalmente nos suele llevar a la página anteriormente visitada, en una aplicación que utilice AJAX puede engañarnos, ya que aún habiendo cambiado completamente la vista, no es necesario haber cambiado de página. Por ello es necesario ponerse en la piel de usuario y ver cuando es coherente hacer el uso de esta herramienta para evitar malas experiencias y confusiones.

En JSF el uso de AJAX consiste en poner una etiqueta '<f:ajax/>' dentro del elemento que queramos que lo lance y asociar a esta los 'id' de los elementos que queremos que se recarguen tras la acción. En casos en los que tengamos que enviar datos de formularios necesarios para realizar la acción, habrá que indicar también que entradas tomar en cuenta para la ejecución.

2.6.3 CSS

CSS (del inglés *Cascading Style Sheets*) es un lenguaje encargado de describir el estilo o apariencia de una pàgina HTML, es decir, forma una serie de reglas sobre cómo cada uno de los elementos deben ser mostrados.

La sintaxis de un fichero CSS consiste en un conjunto de bloques con dos partes: un selector seguido de cada uno de sus atributos con el valor asignado.

- Selector:

  Es una expresión que define aquellos aquellos elementos que serán afectados por los atributos que se especifican. Están formados identificadores, clases o tipos, y símbolos que indican subconjuntos de los mismos, como puede ser '>', indica 'descendiente directo'.

- Atributos:

  Están escritos como un conjunto de parejas clave-valor de la forma '*clave : valor*' separados por comas y entre llaves. Normalmente el valor es numérico y uno de entre varios predefinidos, pero hay caso en los que el atributo puede tomar un valor formado por varios valores, y dependiendo de su orden y cantidad, se interpretan de una u otra manera. También existen ciertas funciones que pueden tomar el lugar de un valor, tomando como tal el resultado de la función.

Con CSS es posible seleccionar a qué páginas afecta cada hoja de estilos individualmente ya que es necesario declarar explícitamente que archivo CSS han de cargarse para cada HTML.

Además, una hoja de estilos no es el único lugar en el que se pueden declarar atributos, sino que en la propia página HTML existe una etiqueta reservada para ello, e incluso en cada elemento individualmente. Esto puede llevar a que para un elemento existan más de un valor para cierto atributo en varios lugares. Es por ello que CSS tiene jerarquizada la asignación de atributos en varios niveles, dándose que los niveles superiores prevalecen sobre los inferiores (excepto para aquellos atributos marcados con la cadena especial '! *important*'.

Los atributos se asignan en el siguiente orden de menos a más prioritario:

- Los que proceden de una hoja de estilos del propio navegador.
- Los que proceden de una hoja de estilos del usuario.
- Los que proceden de una hoja de estilos de la aplicación.
- Los que proceden de la página HTML, en concreto de la zona *<head>*.
- Las asignadas al propio elemento en el atributo '*style'*.
- Las marcadas con la cadena '!important'.

2.6.4 JPA

La '*Java Persistence API'* (mejor conocida como JPA) es una API de persistencia tanto para Java EE como para Java SE. Su finalidades la de simplificar y estandarizar el uso de modelo de persistencia en aplicaciones Java.

JPA nos da la posibilidad de interactuar con bases de datos desde la propia aplicación, usando clases como una interpretación de las tablas y sus relaciones. Esto se consigue mediante una serie de herramientas que la librería nos ofrece:

- Un conjunto de etiquetas que nos dan la posibilidad de asociar una clase y sus atributos con los diferentes atributos de una base de datos así como sus relaciones. Cada una de estas clases es una '*Entity*', una representación directa de una tabla en Java.

- Una API para la ejecución de acciones en base de datos. Contiene operaciones que van desde la inserción de datos hasta la eliminación y la lectura, tanto usando el lenguaje SQL nativo como uno específico introducido por JPA.

- Un lenguaje adaptado basado en SQL llamado JPQL (*Java Persistence Query Language*) que hace uso de las '*Entities*' para poder realizar consultas SQL de manera mucho más intuitivas y a más alto nivel.

3 Diseño

3.1 Casos de Uso

Se van a dividir los casos de uso en dos grandes grupos en referencia a qué parte del sistema afectan cada uno de ellos, pero teniendo en cuenta que el único punto de interacción de los usuarios con el sistema es a través del cliente o aplicación web en este caso.

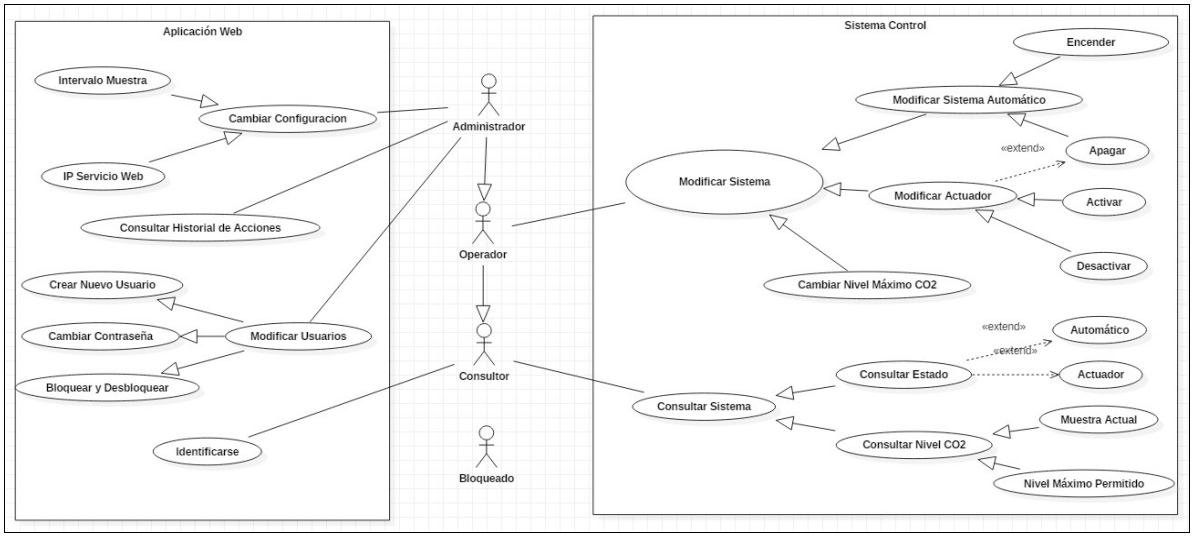

Figura 5. Diagrama de casos de uso.

Los usuarios de la aplicación están separados por perfiles con diferentes roles, siendo cada rol superior una extensión del anterior. Esto es así menos para el rol “Bloqueado” que no puede realizar ninguna acción en el sistema. A continuación, veremos las competencias de cada uno de los perfiles en orden de rol ascendente, entendiendo que las competencias de cada perfil incluyen a las del perfil anteriormente descrito:

- Bloqueado: Cuando un usuario está bloqueado no se le permite realizar ninguna acción en el sistema, ni siquiera iniciar sesión.
- Consultor: Sólo tiene permitido ver tanto el estado del sistema como las muestras tomadas, sin permitir más interacción con este.
- Operador: Además de todo lo que hace el Consultor, el Operador puede realizar todas las acciones posibles referentes al sistema de control. Controlar el sistema automático, el sistema actuador y el nivel máximo de CO_2.
- Administrador: Es el perfil con el rol más alto. Incluyendo todas las funciones del Operador, además tiene la posibilidad de alterar tanto la configuración de la aplicación,

como los distintos usuarios del sistema, teniendo acceso también al registro de acciones realizadas por cada uno de los usuarios.

3.2 Circuito módulo de control

El circuito electrónico del módulo de control esta compuestos tanto por el sensor como por el actuador, así como por la placa Arduino UNO. El diagrama del circuito no es en absoluto complejo, pero sí que es importante tener en cuenta en qué salidas y entradas de Arduino se esperan conectados respectivamente.

En el caso del sensor, el sketch leerá el resultado de las mediciones a través de la entrada analógica 0 (pin A0), mientras que el actuador es accionado a través de la salida digital 8 (pin 8). Además, el sensor precisa de una fuente de 5v para ejecutarse, por lo que es necesario tomar una de las salidas de 5v que ofrece Arduino para alimentarlo.

3.3 Sketch Arduino

El sketch consta de 3 partes, una de inicialización que solo se realiza al inicio, y otras dos que se repiten indefinidamente que son la comprobación del canal de comunicación y el control del nivel de CO2.

1.Inicialización

Durante la inicialización se establecen los valores de las variables y se inicia el canal de comunicación.

2.Comandos

Los comandos reconocidos por el sistema son los siguientes:

- **AUON:** Activa el modo automático.
- **AUOFF:** Desactiva el modo automático.
- **OPACT:** Abre el actuador y desactiva el modo automático.
- **CLACT:** Cierra el actuador y desactiva el modo automático.
- **GETST:** Devuelve el estado del sistema en formato 'Estado Automático'/'Estado Actuador' donde 1 representa Activado y 0 Desactivado para cada uno de los dos. Ej. 1/0 significa 'Automático Activado'/'Actuador Desactivado'
- **GETLVL:** Devuelve el nivel actual de CO2 detectado en ppm.
- **GETMAXLVL:** Devuelve el nivel máximo de CO2 establecido actualmente.

- **SETMAXLVL ‘nivelMaxCo2’:** Establece ‘nivelMaxCo2’ como nuevo nivel máximo ante el cual el sistema se activará.

3.Control del nivel de CO2

Esta fase de la ejecución sólo se ejecutará si está activado el modo automático.

Durante este bloque, el sistema realiza una lectura y compara el nivel de CO2 obtenido con el máximo establecido. En caso de de ser superior la lectura, el sistema bien activa, bien mantiene activo el actuador.

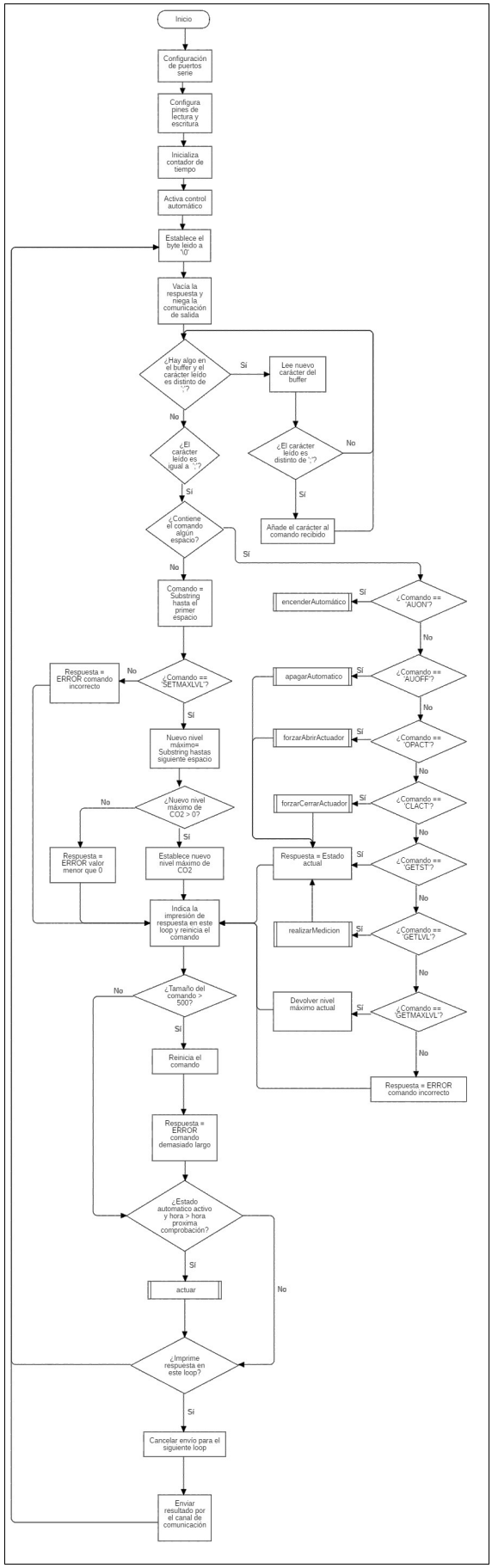

Figura 6. Diagrama de flujo del funcionamiento del sketch de Arduino.

3.4 Servicio Web

El servicio web está dividido en 3 partes: comunicación con Arduino, modelo y API RESTful.

El módulo de comunicación con Arduino contiene una excepción relacionada con este módulo y una clase encargada de gestionar las comunicación mediante la RxTx con Arduino, la clase 'ArduinoSerial'. Contiene una serie de métodos de inicialización así como métodos 'thread safe' para el envío y recepción de información.

El modelo se encarga de mantener ciertos datos de la configuración como el nivel máximo de CO2 y hace de intermediario entre la API y el módulo de comunicación con arduino, de manera que contiene métodos sincronizados que interactúan con el módulo de comunicación, y estos serán llamados desde la API. Todo esto estará englobado en la clase 'RestfulParams', que hace uso de 'ArduinoSerial' en el grueso de sus operaciones.

Por último, declarada en la clase 'Controles', la API contiene los métodos necesarios para gestionar todo el sistema, tanto consulta de estados como acciones del sistema y establecimiento del nivel máximo de CO2. El formato de la respuesta es siempre JSON, lo que lo hace más versátil y estándar a la hora de implementarlo en otros sistemas.

En el paquete 'common' existen dos clases genéricas que albergan valores y funcionalidades comunes a toda la aplicación, y son usadas en general a modo de librería por muchas de las otras clases.

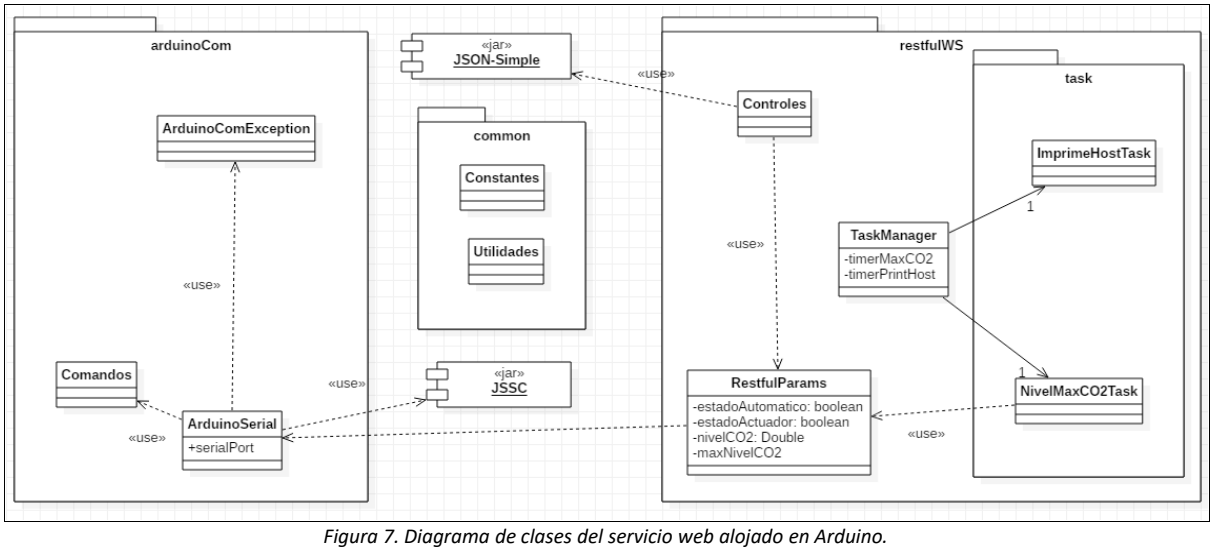

Figura 7. Diagrama de clases del servicio web alojado en Arduino.

3.5 Esquema de Base de datos

El modelo de datos se basa en mantener las muestras y los usuarios del sistema. Por lo tanto, podemos resaltar 3 tablas principales, 'Usuario', 'Acción' y 'Muestra'.

En 'Usuario' guardaremos los diferentes usuarios registrados y sus credenciales, manteniendo la contraseña cifrada. Cada usuario tiene un 'Perfil' asociado que le asigna ciertas competencias dentro del sistema.

'Acción' es la tabla encargada de guardar el registro de las distintas de las distintas acciones que realicen los usuarios, por lo que cada una tiene uno asociado. Además, existen varios tipos predefinidos de acciones, usados para normalizar la búsqueda por tipo.

'Muestra' asocia una medición a una fecha en concreto, manteniendo un registro de estas a lo largo del tiempo. Además, para posibilitar futuras ampliaciones del sistema, se ha creado una tabla 'Tipo_Muestra', en la que se pueden guardar varios tipos de muestras que irán asociadas a cada muestra, pero no es el caso para este proyecto, ya que sólo nos interesa el nivel de CO2.

Por último, la tabla 'Variable' guarda ciertas variables asociadas a la configuración general del proyecto, por lo que serán actualizadas cada vez que uno sean alteradas y serán compartidas por toda la aplicación.

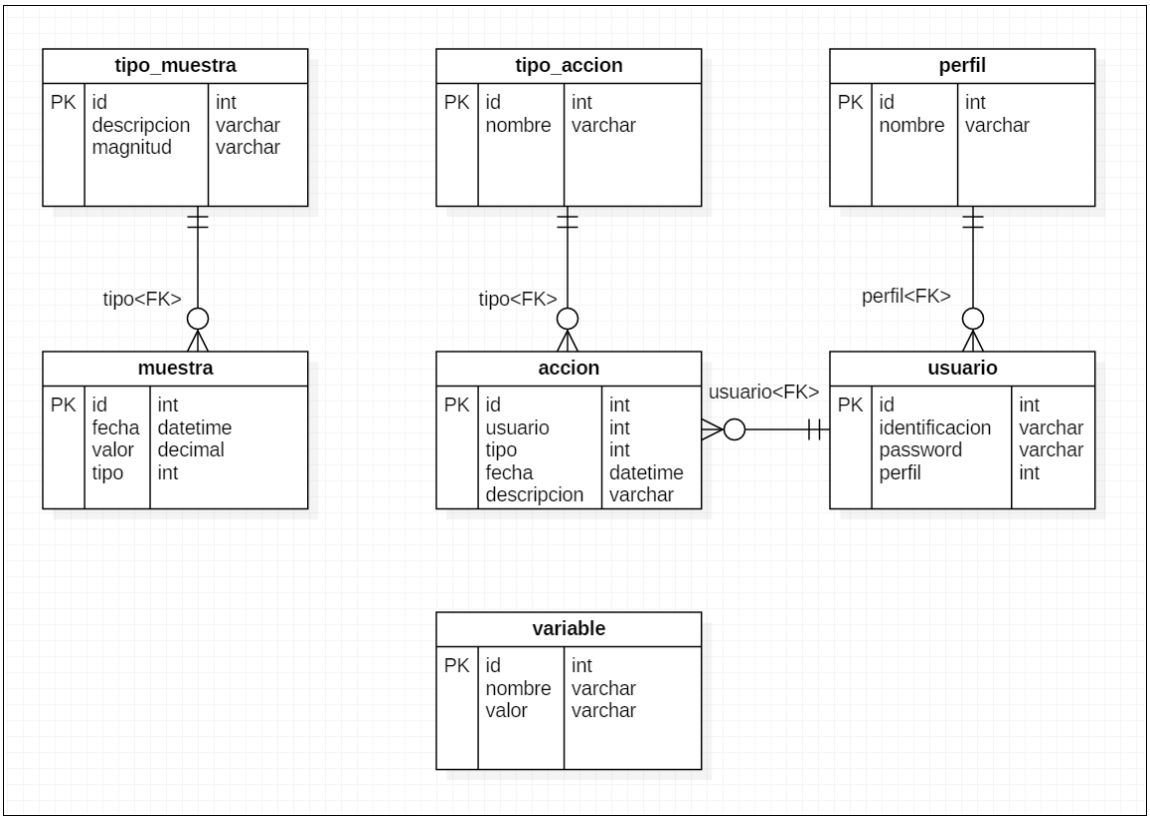

Figura 8. Diagrama Entidad/Relación de las tablas de la base de datos.

3.6 Aplicación Web

La aplicación web consta de 2 partes principales, el panel de control del sistema, y el panel de control de la propia aplicación, en el que se gestionan datos como la configuración de la conexión, el periodo de consulta del nivel de CO2 y los diferentes usuarios así como sus roles.

Este último panel de control es sólo accesible por usuarios con el rol de administrador, por lo que se deberá tener en cuenta a la hora de implementar tanto la vista como el servidor. Además, existe un rol de sólo consulta que no tiene permitido interactuar con el sistema, sino que simplemente puede controlar los niveles y comprobar el estado de este. En este caso le serán bloqueados los controles del sistema.

3.6.1 Interfaz Gráfica

Como se ha comentado, la aplicación dispondrá principalmente de 2 vistas principales, de las cuales sólo una de ellas será accedida por todos los usuarios. Ésta será la pantalla de gestión del sistema (figura 9).

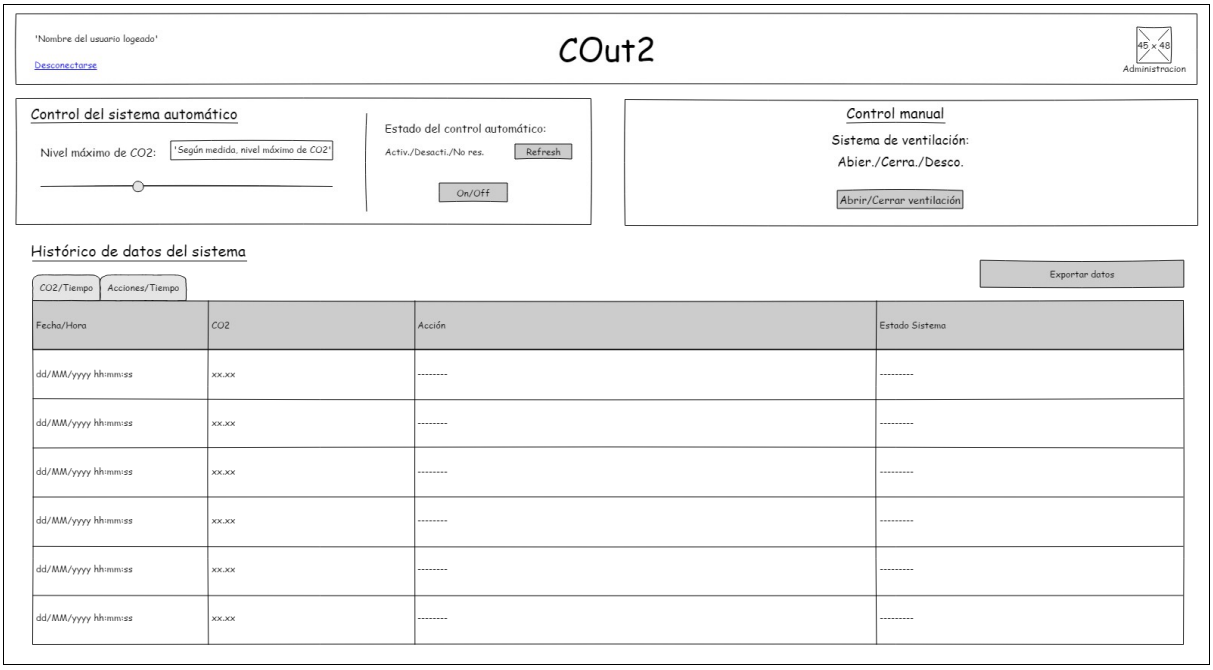

Figura 9. Diagrama de la interfaz gráfica de la aplicación web. Panel de control.

En esta pantalla los usuarios podrán consultar una tabla en la que se verá reflejada la evolución en los niveles de CO2 a lo largo del tiempo, siendo posible filtrar los resultados por la fecha de la muestra. Los botones de interacción con el sistema sólo serán visibles para usuarios con rol 'Operador', lo que permitirá activar/desactivar tanto el sistema automático como el propio actuador, además de establecer el nivel máximo de CO2. Si además, el

usuario es ‘Administrador’, le será visible una lista con las operaciones realizadas por los diferentes usuarios así como la hora en que se realizaron. Además los administradores tendrán acceso a la pantalla de configuración de la aplicación a través del botón situado arriba a la derecha.

El panel de control de la aplicación (figura 10) dispone de una zona de gestión de ciertos parámetros de la aplicación, y otra para a modo de CRU de usuarios. Además, tenemos la opción de bloquear y desbloquear usuarios, cambiar su contraseña de acceso y alterar sus roles.

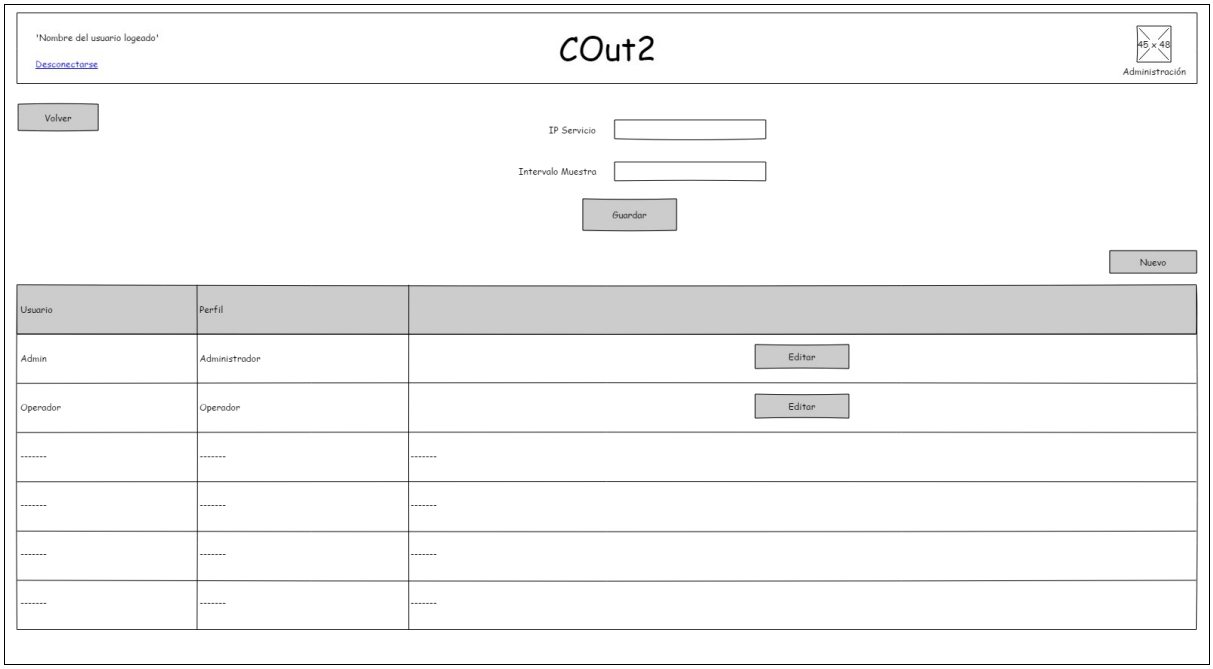

Figura 10. Diagrama de la interfaz gráfica de la aplicación web. Administración de usuarios.

3.6.2 Modelo del Servidor

El modelo de la aplicación está dividido en los módulos referentes a la persistencia, como lo son las entidades y controladores de datos, los beans, clases directamente relacionadas con JSF que hacen de 'controlador' entre la vista y el modelo, y un módulo de clases 'common' que contienen clases utilizadas en numerosos puntos de la aplicación además de dos 'lazyModel' usados para la carga paginada de tablas en la vista.

El módulo JPA está compuesto tanto por las entidades ('entity') como por los diferentes controladores ('controller'). Las entidades representan las distintas tablas de la base de datos y las relaciones entre ella y en el conjunto forman el modelo de datos de la apliucación. Los controladores son usados para realizar operaciones en la base de datos mediante funciones generalmente definidas explícitamente. Todos los controladores

heredan de 'ControllerBase', donde se implementa la conexión con el recurso, cuyo nombre se toma de una variable declarada en 'Constantes'. Es importante para el correcto funcionamiento de la aplicación que ante cambios en el nombre del recurso se actualice correctamente, de lo contrario le será imposible realizar la conexión a los controladores.

'Constantes' y 'Utilidades' de 'commons' son utilizada en la mayor parte de la aplicación, ya que contienes datos y funciones de uso general. Además, 'Popup' estandariza una llamada a un popup de información común para todas las vistas, de manera que puede ser llamado con un simple método estático. El diagrama no refleja dichos uso por la limpieza del mismo, pero cabe resaltar que su finalidad es evitar la repetición de código y la centralización de ciertas configuraciones (como se ha visto en el caso del nombre del recurso).

Por último, el módulo 'beans' es el mas extenso de todos. Existen 4 beans importantes que o bien representan una parte importante de la aplicación o bien una vista completa. Esto es porque dentro de las dos vistas principales existen beans para ciertas partes con funcionalidades concretas, como pueden ser 'CrearUsuario', utilizado exclusivamente en el 'Popup' de creación de usuarios, 'Controles' para toda la interacción con el sistema automático o 'GraficaBean' para la generación de la gráfica de muestras.

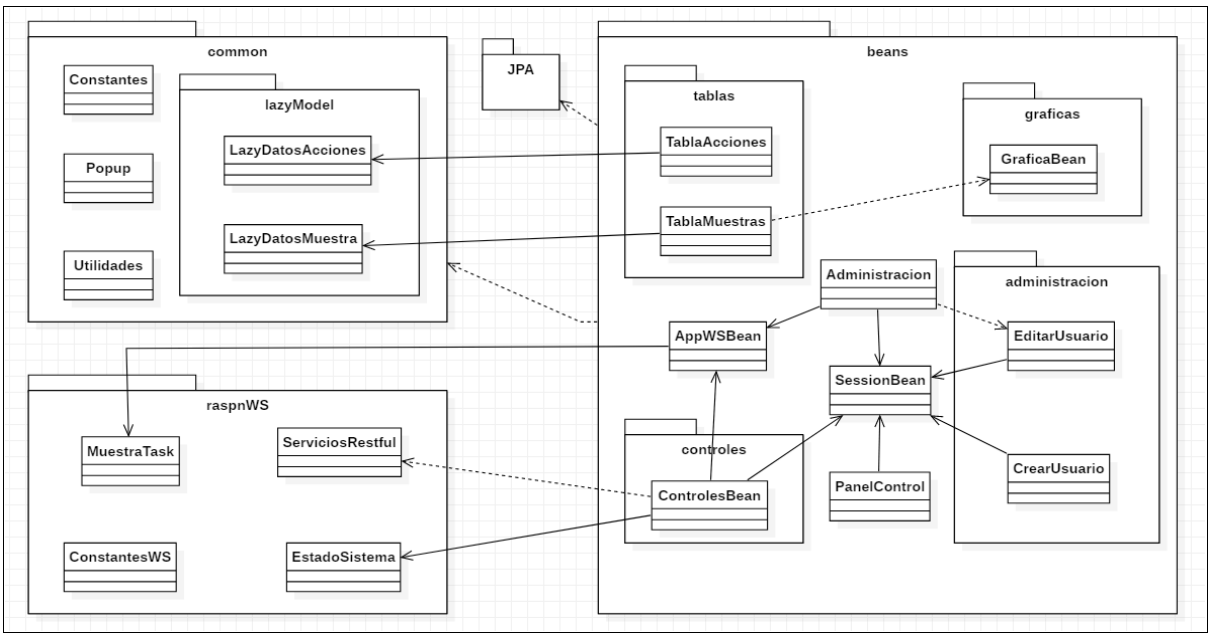

Figura 11. Diagrama de clases de la aplicación web.

Los 4 beans más importantes son los siguientes:

- **AppWSBean:** Un bean de aplicación encargado de gestionar las variables de configuración de la aplicación (la url de conexión con el servicio web así como en intervalo entre la toma de muestras) además de controlar un TimerTask encargado de la toma de muestras.
- **SessionBean:** Un bean de sesión encargado exclusivamente del inicio de sesión, manteniendo el usuario conectado y de desconexión. Este bean es usado regularmente para asegurar, desde otros puntos de la aplicación, por qué usuario se están realizando las distintas operaciones, de manera que se toma dicho usuario y se guarda la operación asociada en la base de datos.
- **PanelControl:** Es usado por la vista principal. Su funcionalidad principal es la de establecer qué elementos de dicha vista son mostrados y cuales no.
- **Administracion:** Este bean es usado solo en el panel de administración accesible únicamente por los usuarios con perfil 'Administrador'. Tiene funciones como iniciar la creación de un usuario y su edición, para bloquear y desbloquear usuarios y para la modificación de algunas configuraciones de la aplicación.

4 Desarrollo

4.1 Sketch Arduino

El sketch, según el modelo, consta de 2 funciones principales: la lectura e interpretación de comandos, y el control del nivel. Ambas partes se han desarrollado sin problemas, encontrando

4.1.1 Lectura e implementación del canal de comunicación

En cada ciclo se comprueba si hay algún nuevo carácter en el buffer. En caso afirmativo, añade ese carácter a una cadena acumulativa que va almacenando el comando completo. El 'punto y coma' (;) significa el fin del comando, y cuando es encontrado, se toma la cadena y se actúa en función del comando recibido.

```
while(Serial.available() > 0 && char(inByte) != ';'){
  inByte = Serial.read();
  if(char(inByte) != ';')
    command += char(inByte);
}
```

Figura 12. Lectura de carácter desde la comunicación serie.

Si el comando recibido no es ninguno de los reconocidos, se devuelve un error con dicho mensaje. En caso de llegar a 500 caracteres, se reinicia el comando y devuelve un error.

```
if(command.length() > 500){
  command = "";
  Serial.println("ERROR: COMANDO DEMASIADO LARGO. REINICIANDO CADENA");
}
```

Figura 13. Reinicio del comando leído al llegar a los 500 caracteres.

4.1.2 Control automático

Además de diferentes constantes como el el tiempo entre tomas de de muestras, los diferentes pins y alguna variable auxiliar (como puede ser 'inByte', donde se guarda cada byte leído, el programa basa su comportamiento en 3 variables principales, 'autoState', 'actState' y 'proximaComprobacion'.

- **autoState**: Se encarga de mantener el estado del sistema automático. Esta variable es consultada en cada ciclo para saber si el control del aire debe actuar o si, por el contrario, está activado el modo manual.

- **actState:** Se encarga de mantener el estado del actuador. Al contrario que es anterior, no es consultado regularmente y su función principal es la de mantener la coherencia en las respuestas que se dan a aquellos comandos que devuelven el estado actual del sistema.

- **proximaComprobacion:** Guarda el momento a partir del cual comprobar de nuevo el nivel de CO2 (siempre y cuando el sistema automático esté activado). Cuando el sistema automático completa un ciclo se incrementa esta variable para saber cuando debe realizar el siguiente.

Cabe destacar el simple algoritmo que se ha usado para gestionar la acción del actuador constando de una toma de muestra y la decisión de apertura que, en caso de tomar una lectura mayor del máximo establecido, actuará activando el ventilador por un ciclo de 1000 ms y en caso de mantenerse los niveles por debajo del máximo, sólamente esperará 300 ms. De esta manera se tiene un control del nivel mucho más exhaustivo manteniendo el nivel por debajo del máximo en el mayor número de casos.

Es importante aclarar que en caso de estar desactivado el sistema automático, esta comprobación se omite.

4.2 Ajuste del sensor

Como hemos comentado, el sensor es una parte fundamental de este proyecto, ya que sobre él se recae una de las funcionalidades principales de este proyecto, la lectura del nivel de partículas contaminantes en el ambiente en proporción al total, medido en 'ppm' o 'partes por millón'. Sin embargo es un aparato electrónico y su capacidad para transmitir información está limitada a la transmisión de un voltaje que represente los valores obtenidos.

Es por ello que saber interpretar los resultados es una de las primeras tareas a cubrir, y para ello necesitamos una fuente de información donde se indique de qué manera entender las lecturas del sensor. Esta fuente de información existe de manera muy común para este tipo de componentes y es lo que llamamos el '*datasheet*'. El *datasheet* es, como su nombre indica, una hoja de datos que define las especificaciones técnicas de un componente. En el caso de nuestro sensor, su *datasheet* (Anexo a) nos indica sus condiciones de funcionamiento en condiciones normales, como se comporta respecto a cambios en el ambiente y las lecturas estimadas según los niveles de distintos gases. Esto último es fundamental para averiguar la función sobre la que calcular los niveles en base al voltaje entregado, y está representado en la siguiente gráfica (figura 14).

Pero, ¿qué nos dice exactamente esta gráfica? Pues nos indica, bajo unas condiciones de

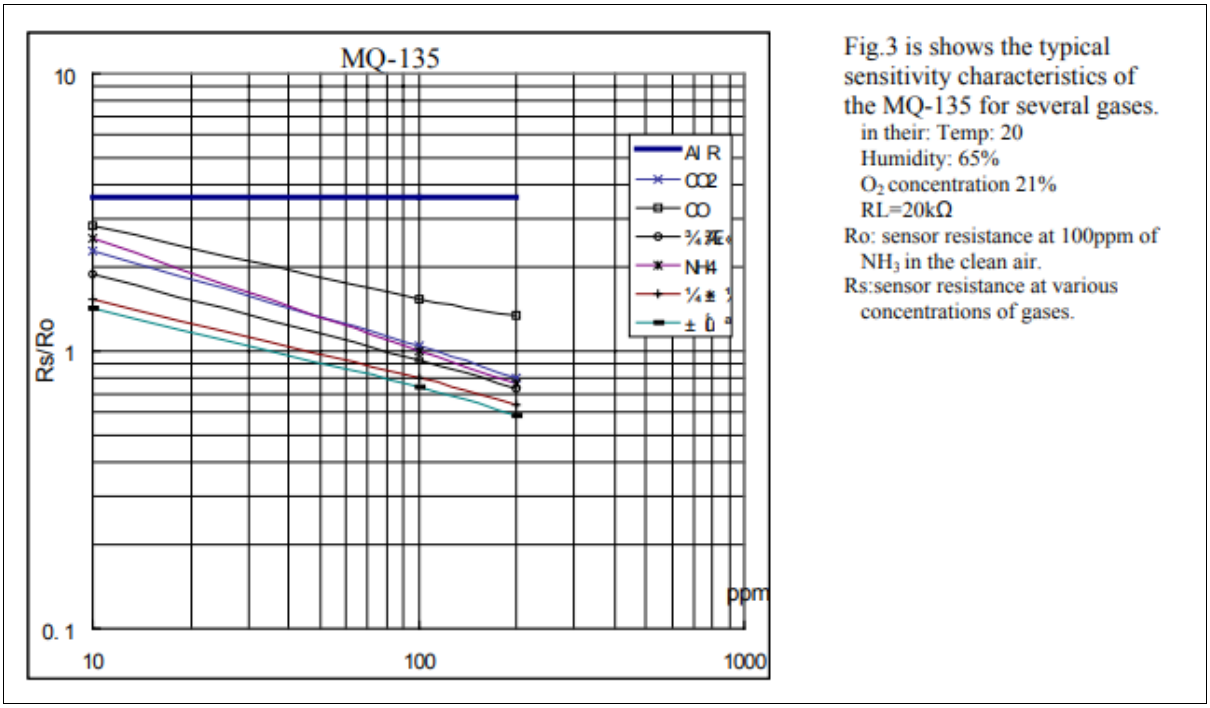

Figura 14. Relación de los valores 'Rs/R0' y los ppm de diferentes elementos. Extraído del Datasheet del MQ-135 (Anexo A).

temperatura, humedad y concentración de oxígeno concretas, la relación entre los ppm obtenido (separados por cada gas) y 'Rs/R0'. 'Rs' es la resistencia variable que ofrece el sensor según la diferente cantidad de gases detectados, y por lo tanto es el valor a tener en cuenta como variable en nuestra ecuación. Pero aún falta otro factor en esta función, 'R0'. 'R0' es un valor de referencia del sensor, pero no nos dan un valor constante en el *datasheet*. Esto es porque no es posible, ya que 'R0' es un valor que varía dependiendo de cada sensor individualmente, y representa el valor de la resistencia ante 100 ppm de NH3

(Amoníaco) bajo las condiciones dichas anteriormente. Pero, como hemos dicho, el sensor nos entrega un voltaje, y de él debemos inducir la resistencia, ¿cómo hacemos esto?

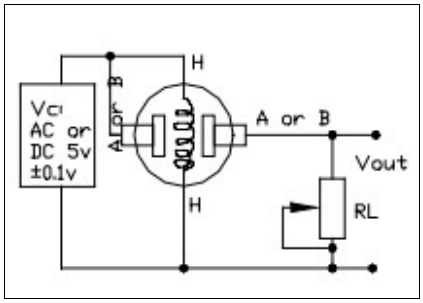

Figura 15. Circuito del MQ-135. Extraído del Datasheet del MQ-135 (Anexo A).

Para averiguarlo, vamos a basarnos en la siguiente explicación[5] para, usando la ley de Ohm, calcular dicha resistencia. En ella se usa como referencia un sensor MQ-4 en lugar de nuestro MQ-135, pero el funcionamiento es similar para ambos, por lo tanto así vamos a calcular 'Rs':

Según la ley de Ohm

$$I = V / R$$

Y como podemos ver, las resistencias 'Rs' (Definida por los extremos A y B) y 'RL' (resistencia variable) están conectadas en serie (figura 15), por lo que en la ley de Ohm se resumen en una suma de resistencias, quedando lo siguiente:

$$I = Vc / (Rs + RL)$$

Por lo que uniendo esto con lo que podemos derivar del circuito completo:

$$V = I \times R$$

$$Vout = I \times RL$$

Teniendo la anterior igualdad:

$$Vout = [Vc / (Rs + RL)] \times RL$$

$$Vout = (Vc \times RL) / (Rs + RL)$$

Bien, ahora debemos tener en cuenta que el valor que nos interesa es 'Rs', y que nuestra variable en esta ecuación debe ser 'Vout', ya que esta es la lectura que recibiremos. Pues el siguiente paso va a ser despejar 'Rs' de la ecuación:

$$Vout \times (Rs + RL) = (Vc \times RL)$$

$$(Rs + RL) = (Vc \times RL) / Vout$$

$$\mathbf{Rs = [(Vc \times RL) / Vout] - RL}$$

Con esto tendríamos la función necesaria para calcular nuestra 'Rs' ya que el resto de términos o son variables (como lo es 'Vout') o son valores definidos:

- Vc es la tensión de entrada, 5 V
- RL es la resistencia variable, que según el *datasheet* recomienda establecer a 20 KΩ (entre 10 KΩ y 47 KΩ), pero el propio sensor lleva ya incluida una de 1KΩ, por lo que habrá que añadirle 20KΩ más.

Por lo tanto, quedaría finalmente:

$$\mathbf{Rs = [(5\ V \times 21000\ \Omega) / Vout] - 21000\ \Omega}$$

Y con esto tendríamos definitivamente 'Rs'. Ahora que sabemos 'Rs' y 'R0' es calculable (lo veremos más adelante) nos falta encontrar la función representada en el *datasheet* que nos da la relación entre (Rs/Ro) y ppm. Pues para esto necesitaremos ver y entender la gráfica de comportamiento del sensor frente a los gases, y desde esta inferir dicha función.

En primer lugar, vemos una función aparentemente lineal, pero si nos fijamos cuidadosamente, los ejes están deformados, habiendo más separación entre unos puntos que entre otros. Esto se debe a que la función no es lineal, si no que la gráfica está deformada para representarla linealmente, pero, si normalizamos los ejes, sigue la forma de una función logarítmica. Por lo tanto, hay que tenerlo en cuenta para sacar la función correctamente.

Empezaremos tomándolas como lineales en primer lugar. Las ecuaciones lineales tienen la siguiente forma:

$$y = mx + b$$

Donde 'm' es la pendiente y 'b' la intersección con el eje Y. Sabiendo esto y que nuestra función original sigue una escala logarítmica, la fórmula real debería tener la siguiente forma:

$$\log(y) = m*\log(x) + b$$

Por lo que lo primero que vamos a calcular es la pendiente (‘m’), que se calcula aplicando la siguiente fórmula (siendo (x1,y1) y (x2,y2) dos puntos conocidos de la gráfica):

$$m = [\log(y2) - \log(y1)] / [\log(x2) – \log(x1)]$$

$$m = \log(y2/y1)/ \log(x2/x1)$$

Según la gráfica, dos puntos conocidos podrían ser por ejemplo (10,2.3) y (200,0.8) respectivamente, por lo tanto nuestra pendiente surgirá de la siguiente ecuación:

$$m = \log(0.8/2.3)/ \log(200/10)$$

$$m = \log(0.34783)/\log(20) = \mathbf{-0.352519}$$

Por último, para definir ‘b’, solo debemos tomar otro punto de la gráfica y sustituir los valores en la función, además de ‘m’ que ya la tenemos. Por ejemplo el punto (10.2.3):

$$\log(2.3) = -0.352519*\log(10) + b$$

$$b = \log(2.3) + 0.352519 = \mathbf{0.714247}$$

Aplicando estos dos valores a la función tenemos:

$$\log(y) = -0.352519*\log(x) + 0.714247$$

Que adaptado a nuestro ejemplo particular donde el eje Y es Rs/R0 y el eje X son los ppm, que es lo que nos interesa despejar, nos queda aproximada la función de la siguiente manera:

$$\log(Rs/R0) = -0.352519*\log(PPM) + 0.714247$$

$$\log(Rs/R0) - 0.714247 = -0.352519*\log(PPM)$$

$$[\log(Rs/R0) - 0.714247] / -0.352519 = \log(PPM)$$

PPM = 10^{[log(Rs/R0) - 0.714247] / -0.352519}

Pero aún nos falta averiguar ‘R0’, la referencia, para terminar de ajustar nuestro sensor. Esto requiere de un equipo muy costoso para realizarlo de manera precisa, y se realiza tomando la referencia de 100 ppm de Amoníaco que nos dan y ajustando R0 hasta obtener dicho

valor, tomando como referencia un sensor que ya esté configurado y cuyas lecturas sean correctas.

Ya que los costes son excesivos y la razón de este proyecto es la implementación del sistema completo en sí y no la precisión de sus mediciones (ya que se mueve dentro de un rango seguro y no en niveles críticos) se ha estimado innecesario el gasto material y de esfuerzo para dicho ajuste, por lo que se tomarán valores aproximados perfectamente válido. Por lo tanto, sabiendo que los ppm de CO2 en el aire rondan los 360 ppm en aire limpio y 700 ppm en ciudad, tomaremos un valor que se mueva alrededor de los 500 ppm. En el caso de este sensor, el valor de R0 parece estar sobre los 250000 Ω, por lo que la fórmula nos quedaría:

PPM = 10^{[log(Rs/200000) - 0.714247] / -0.352519}

Por último, al implementar esta fórmula en Arduino, nos ha quedado la función como en la figura 16.

```
double realizarMedicion(){
  lectura = analogRead(PIN_IN_CO2);
  float vAux = lectura * (5.0 / 1023.0);
  float Rs=(105000.0/vAux)-21000;
  double co2 = pow(10, (log10(Rs/R0)-0.714247)/(-0.352519));
  return co2;
}
```

Figura 16. Cálculo del nivel de CO2 en el sketch de Arduino.

4.3 Servicio Web

El servicio web se divide en tres módulos principales, la declaración y funcionamiento servicios RESTful, una clase que mantiene los datos guardados por el servicio referentes al estado del sistema de control, lo que llamaremos es modelo, y la clase que gestiona la comunicación con Arduino. Además, y aunque no es un módulo propiamente, existen dos tareas periódicas importantes para un mejor funcionamiento del sistema que se comentarán más adelante.

4.3.1 Servicios

Se han desarrollado 8 servicios diferentes para el control del sistema. Están todos desarrollados en la clase '*Controles*'.

Los servicios son los siguientes:

- **getEstadoActual:** Devuelve el estado actual del sistema.
- **activarAutomatico:** Activa el sistema automático en Arduino y devuelve el estado actual del sistema.
- **desactivarAutomatic:** Desactiva el sistema automático en Arduino y devuelve el estado actual del sistema.
- **activarActuador:** Activa el actuador y desactiva el sistema automático en Arduino y devuelve el estado actual del sistema.
- **desactivarActuador:** Desactiva el actuador y el sistema automático en Arduino y devuelve el estado actual del sistema.
- **getMuestra:** Devuelve una muestra actual CO2.
- **getNivelMax:** Devuelve el nivel máximo de CO2 establecido actualmente.
- **setNivelMax:** Establece un nuevo nivel máximo de CO2 y devuelve el estado actual del sistema.

Cada uno de estos servicios realizan 2 tareas: derivar la tarea y capturar los datos a enviar, empaquetándolos los datos en formato JSON para devolverlos como resultado. Por lo tanto, todos hacen de meras APIs para que hacen llamadas a la clase '*RestfulParams*'.

A la hora de preparar la respuesta, hemos de crear un texto con el formato JSON, y para ello se ha elegido usar la librería JSON-Simple (Anexo b). Es una librería sencilla de usar de código abierto y su funcionamiento es el siguiente:

Se crea un objeto "*JSONObject*".

JSONObject obj = new JSONObject();

Se le añaden pareja clave-valor como si de un mapa se tratase con '*put*'.

obj.put("key",value);

Y por último, 'toJSONString' transforma el objeto al formato JSON.

obj.put.toJSONString();

Por último, comentar la función '*estadoSistema*' que se encarga de obtener el estado del sistema desde '*RestfulParams*', ya que como era necesario para la mayoría de servicios, era preferible declarar una función con dicho objetivo.

```
52      @Path(Constantes.GET_ESTADO_SISTEMA)
53      @GET
54      @Produces("application/json")
55      public String getEstadoActual(){
56
57          RestfulParams restfulParams = RestfulParams.getRestfulParams();
58
59          return estadoSistema(restfulParams);
60      }
```

Figura 17. Ejemplo de declaración de un servicio. Servicio 'getEstadoActual'.

4.3.2 Modelo del estado

El modelo de datos de esta aplicación se reduce a una representación del estado del sistema en diferentes variables, todas ellas declaradas en la clase '*RestfulParams*'.

Estas variables son la siguientes:

- **estadoAutomatico:** Guarda si el sistema automático está o no activado.
- **estadoActuador:** Guarda si el actuador del sistema está o no activado.
- **maxNivelCO2:** Guarda el nivel máximo de CO_2 establecido actualmente.
- **nivelCO2:** Guarda la última muestra que se tomó del nivel de CO_2 por el sistema.

'*RestfulParams*' hace de intermediario entre '*ArduinoSerial*', la clase encargada de comunicarse con Arduino, y la ya comentada '*Controles*'. Por motivos principalmente de integridad en los datos, sigue el patrón Singleton[6], de esta manera evitamos dar respuestas diferentes ante diferentes peticiones simultáneas, ya que no sólo hay una fuente de datos, y además permite un control del flujo de peticiones para las acciones más concurridas, como lo es la consulta del estado, aplicando un tiempo de espera entre consultas de este tipo de manera que si se excede el número de estas, se devuelva el estado guardado el las variables anteriores.

Para mantener la configuración ante un posible reinicio del servicio, se mantiene una un archivo de texto con el último '*maxNivelCO2*' establecido, de manera que al cargar de nuevo el servicio se busca la existencia de dicho archivo para leer mantener el valor. Esto está pensado para una posible mejora en la que se tengan más opciones, ya que el archivo

guarda los datos en formato JSON, por lo que es posible recuperar y modificar todas las opciones que necesiten.

4.3.3 Conexión con Arduino

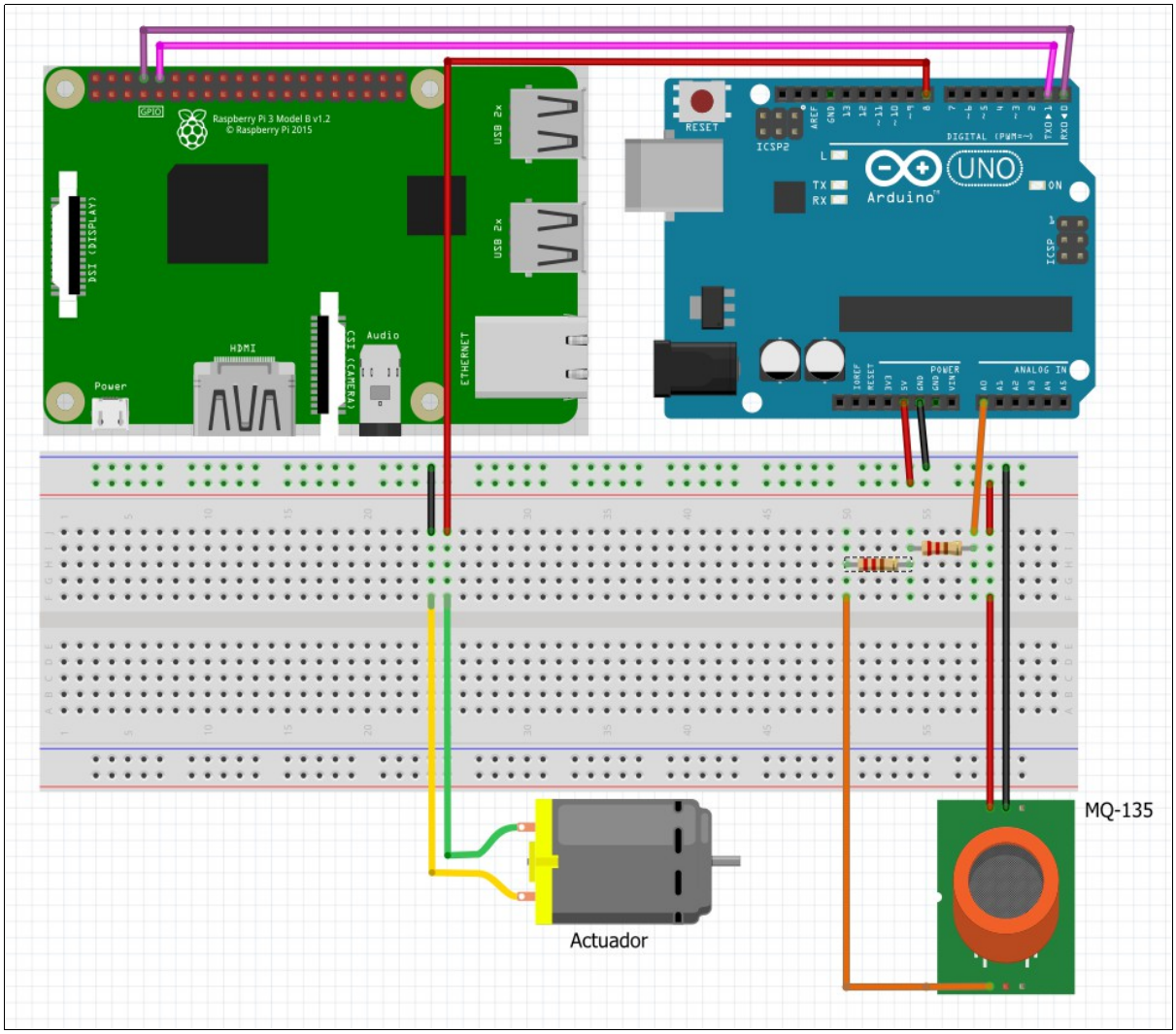

Figura 18. Circuito completo incluyendo la conexión entre Arduino y Raspberry. Elaborado con Fritzing.

Para la conexión con Arduino era necesario encontrar un método de acceder al puerto serie RxTx. Tras buscar alguna herramienta que facilitara esta tarea, se encontró la librería de código JSSC (*Java Simple Serial Connector. A*nexo c). Esta librería simplifica la conexión por un puerto serie en Java, ya que ofrece clases con funcionalidades como la detección de diferentes puertos disponibles o la configuración e inicio de una comunicación mediante dicho puerto. En el siguiente fragmento de código (figura 19) vamos a analizar cómo se realiza la conexión en nuestra aplicación.

```
32      private static void initConexion() throws SerialPortException{
33          String[] puertosDisponibles = SerialPortList.getPortNames();
34
35          if(puertosDisponibles.length < 1){
36              throw new ArduinoComException("No hay puertos disponibles.");
37          }else{
38              int i = 0;
39              boolean creado = false;
40              while(i<puertosDisponibles.length && !creado){
41                  if(puertosDisponibles[i].matches(".*/ttyS.")){
42                      serialPort = new SerialPort(puertosDisponibles[i]);
43                      creado = true;
44                  }
45                  i++;
46              }
47          }
48
49          synchronized(serialPort){
50              serialPort.openPort();
51              serialPort.setParams(SerialPort.BAUDRATE_38400,
52                                   SerialPort.DATABITS_8,
53                                   SerialPort.STOPBITS_1,
54                                   SerialPort.PARITY_NONE);
55          }
56      }
```

Figura 19. Procedimiento de inicialización de puerto serie.

En primer lugar usamos '*SerialPortList.getPortList*' para obtener una lista de los puertos disponibles. Esto nos devuelve un array de *String* con todos los puertos disponibles. En esta lista, iteramos todos los nombre buscando uno que termine con la cadena "/ttyS" y un número (generalmente es 0), ya que esta es la forma en la que los puertos que nos interesan son denominados, y se selecciona el primero que cumpla dicho patrón. Con su denominación, se crea un nuevo '*SerialPort*'.

Se abre el puerto para iniciar la conexión y se realiza la configuración. Esta configuración se realiza mediante '*SerialPort.setParams*', y en nuestro caso particular se establece la siguiente configuración:

- Un baudrate de 38400
- 8 bits de datos
- 1 bit de parada
- Ningún bit de paridad

Esta configuración obedece a una de las configuraciones de comunicación serial más comunes, la llamada 8N1 (*8 bits, no parity, 1 stop bit*) aunque por defecto se suele usar un

baudrate de 9600. Esta configuración es muy importante ya que debe coincidir exactamente con la de Arduino, y en caso de no ser así dará lugar una comunicación no exitosa. Estos son solo algunos de los parámetros de configuración de una comunicación serie[7]. Una vez tenemos el puerto abierto y configurado estamos listos para enviar y recibir información. Ya que es una tarea que usan varios métodos, se ha optado por realizar el método '*mandarComando*'.

```
58      private static synchronized String mandarComando(String operacion, String[] argumentos){
59          String comando = operacion;
60          String result = null;
61
62          if(comando != null){
63              if(argumentos != null){
64                  for(int i = 0; i < argumentos.length ; i++){
65                      if(argumentos[i] != null){
66                          comando += ' '+argumentos[i];
67                      }
68                  }
69              }
70              comando += ';';
71
72              byte[] inBuffer = null;
73
74              try{
75                  serialPort = getSerialPort();
76
77                  synchronized(serialPort){
78                      try {
79                          byte[] auxBuff;
80                          String auxResult = "";
81                          serialPort.purgePort(SerialPort.PURGE_TXCLEAR);
82                          long sendMillis = System.currentTimeMillis();
83                          serialPort.purgePort(SerialPort.PURGE_RXCLEAR);
84                          inBuffer = serialPort.readBytes();
85                          serialPort.writeBytes(comando.getBytes());
86                          while ((result == null || serialPort.getInputBufferBytesCount() > 0) && System.currentTimeMillis()-sendMillis < 5000) {
87                              if(serialPort.getInputBufferBytesCount() > 0){
88                                  auxBuff = serialPort.readBytes();
89                                  auxResult += new String(auxBuff);
90                              }
91                          }
92                          inBuffer = auxResult.getBytes();
93                      } catch (SerialPortException ex) {
94                          Logger.getLogger(ArduinoSerial.class.getName()).log(Level.SEVERE, null, ex);
95                      }
96                  }
97                  if(inBuffer == null){
98                      Logger.getLogger(ArduinoSerial.class.getName()).log(Level.SEVERE, null,new ArduinoComException("No hubo respuesta desde Arduino."));
99                  }else{
100                     result = new String(inBuffer);
101                     if("".equals(result))
102                         result = "ERROR: Error respuesta vacía.";
103                 }
104             }catch(SerialPortException ex){
105                 Logger.getLogger(ArduinoSerial.class.getName()).log(Level.SEVERE, "ERROR: Error al creal el SerialPort");
106                 result = null;
107             }
108         }
109         return result;
110     }
```

Figura 20. Procedimiento 'mandarComando'.

Mandar comando, como su propio nombre indica, toma un comando que queramos enviar por el puerto serie y lo envía a través del puerto. Toma un *String* que representa al comando y un array de *String* que serán los posibles argumentos y los une separados por espacios, añadiendo un ';' al final de toda la cadena. Después se transforma toda la cadena en un array de *byte* y se envía por el *buffer* ('*serialPort.writeBytes*'). Después espera una respuesta por un máximo de 5 segundos y, en caso de no obtenerla, se lanza un error advirtiéndolo. Si todo va bien, se eleva la respuesta.

El canal de comunicación es mediante un puerto serie compartido por todos los procesos y, por lo tanto, hay que seguir unas pautas para controlar el acceso en un cierto orden y así

evitar problemas. Por ello, la clase '*ArduinoSerial*' toma una serie de medidas para limitar el número de accesos a uno al mismo tiempo, que es básicamente usar synchronized tanto en los métodos comunes como en el propio '*serialPort*'.

4.3.4 Tareas periódicas

Por último, comentar 2 tareas que se realizan periódicamente en segundo plano. Estas se inician en la clase 'TaskManager'. Para ambas se ha utilizado un '*TimerTask*', que con ayuda de un '*Timer*', permite fácilmente establecer un proceso que se ejecute de forma periódica.

```
34      @PostConstruct
35      public void init() {
36          if(timerMaxCO2 == null){
37              timerMaxCO2 = new Timer();
38              timerMaxCO2.scheduleAtFixedRate(new NivelMaxCO2Task(), 5000, PERIODO_MAX_CO2_TASK);
39          }
40
41          if(timerPrintHost == null){
42              timerPrintHost = new Timer();
43              timerPrintHost.scheduleAtFixedRate(new ImprimirHostTask(), 5000, PERIODO_PRINT_HOST_TASK);
44          }
45      }
```

Figura 21. Programación de tareas periódicas en 'TaskManager'.

ImprimirHostTask: La idea de esta tarea es imprimir en consola periódicamente el *host* en el que está lanzado el servicio. De esta manera, ante cierta duda o simplemente por estar seguro de que estamos conectándonos de manera correcta, esta información será accesible rápidamente.

NivelMaxCO2: Esta tarea es muy importante, ya que responde a solventar una posible situación en la que Arduino se reinicie. Arduino dispone de una memoria permanente, pero esta es muy volátil y no es recomendable contar con ella si se van a realizar muchas operaciones de lectura y escritura. Por ello, si en algún momento perdiera la conexión a la corriente y tuviera que reiniciarse, perdería la configuración que, en este caso, es el nivel máximo de CO2. Por ello, el servicio es quién se encarga de controlar que la configuración sea la esperada en todo momento, y esta tarea cumple esta función muy simple. Cada cierto periodo de tiempo, se toma el nivel máximo establecido y, si es diferente al que se espera, lo establece al adecuado.

4.4 Aplicación Web

Este el último módulo desarrollado de todo el sistema. Tiene 3 objetivos principales: permitir la gestión de usuarios, la interacción con el sistema control y la recolección de datos basado en las lecturas del sistema. Además de esto, se deberá desarrollar la base de datos que utilizará y desplegará en un servidor de bases de datos fácilmente accesible por el servidor de aplicaciones.

4.4.1 Base de Datos

Tras instalar MySQL server, iniciamos Workbench y guardamos una nueva conexión. Para esto, necesitamos saber la dirección IP y el puerto donde está desplegado el servidor. En este caso, el servidor va estar en el mismo equipo, por lo que la dirección será "*localhost*", y el puerto será el puerto por defecto para MySQL '3306'. Por último, introducir las credenciales de administrador.

Una vez realizada la conexión, Workbench ofrece una interfaz para el desarrollo de la estructura de la base de datos del que se ha hecho uso para plasmar los modelos de la fase de diseño.

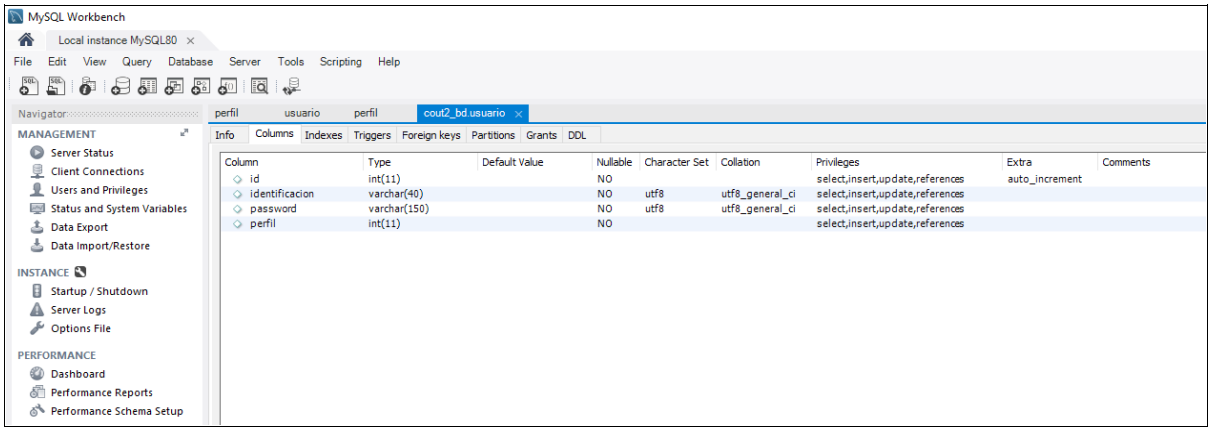

Figura 22. Ejemplo de diseño mediante la interfaz gráfica de la tabla 'Usuario'.

Una vez se ha terminado de desarrollar la base de datos, hay que hacer referencia a esta en el servidor de aplicaciones creando un pool de conexión. En este caso usamos Glassfish y esto se puede en la consola de administrador. Pulsando 'JDBC Connection Pools' → 'New...' y en las propiedades adicionales rellenar los campo Url', 'Password' y 'User'. En la figura 23 vemos como queda en nuestro caso.

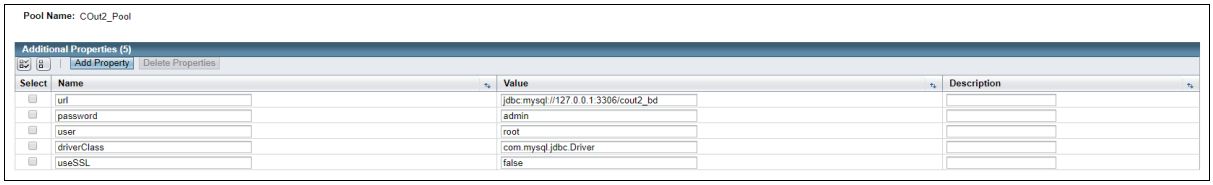

Figura 23. Propiedades adicionales del pool de conexiones.

Además, queremos que desde la aplicación podamos acceder a dicho pool, para lo que es necesario crear lo que se llama un "recurso". Para esto pulsamos 'JDBC Resources' → 'New...' y seleccionamos el pool al que hacemos referencia y el nombre del recurso.

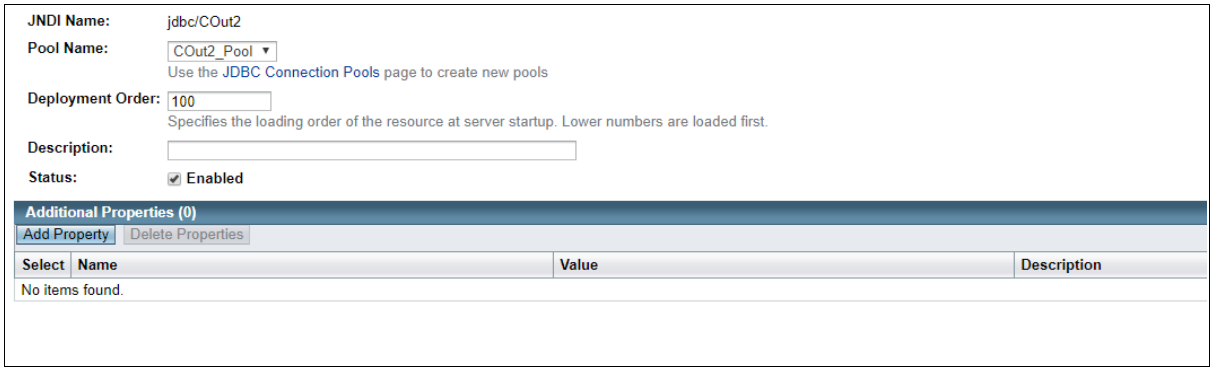

Figura 24. Recurso JDBC declarado en Glassfish.

Una vez tenemos listo el recurso, desde el IDE Netbeans podemos importar el modelo de base de datos usando JPA, con lo que ya tendríamos las distintas clases Java como representación de las distintas tablas para poder usarlas en nuestra aplicación.

4.4.2 Comunicación con el Servicio Web

Al ser RESTful, las llamadas al servicio se hacen mediante simples llamadas HTTP. Todas estas llamadas están definidas en la clase '*ServiciosRestful*', y hacen uso de la función '*llamadaServicio*' (figura 25), un método que toma el servicio, el método del servicio y, en caso de ser una llamada *GET* y necesitase, un mapa de parámetros. Dicha función concatena la URL donde está alojado el servicio, el servicio y el método en concreto, le añade los parámetros según sea necesario, y envía una petición HTTP, recupera la respuesta y devuelve el resultado en texto plano.

```
144     private static String llamadaServicio(String servicio, String metodo, MetodoRequest metodoRequest, Map<String,Object> parametros){
145         String result = null;
146 
147         try{
148             Object auxBean = Utilidades.buscarBean("appWSBean");
149             AppWSBean appWSBean = auxBean == null ? null : (AppWSBean) auxBean;
150             Variable urlVar = appWSBean == null ? null : appWSBean.getUrlRaspberry();
151 
152             if(urlVar == null){
153                 urlVar = new VariableC().findVariableByNombre(Constantes.VARIABLE_URL_RASPBERRY);
154             }
155 
156             String urlString = (urlVar == null ? Constantes.POR_DEFECTO_URL_RASPBERRY : urlVar.getValor())+ConstantesWS.APLICACION_RASPN+servicio+metodo;
157             if(metodoRequest.equals(MetodoRequest.GET) && parametros != null){
158                 urlString += "?"+parametrosGet(parametros);
159             }
160 
161             URL url = new URL(urlString);
162             HttpURLConnection con = (HttpURLConnection) url.openConnection();
163             con.setRequestMethod(metodoRequest.toString());
164 
165             try(BufferedReader in = new BufferedReader(new InputStreamReader(con.getInputStream()))){
166                 String inputLine;
167                 StringBuffer content = new StringBuffer();
168                 while ((inputLine = in.readLine()) != null) {
169                     content.append(inputLine);
170                 }
171                 in.close();
172                 result = content.toString();
173             }catch(Exception e){
174                 Logger.getLogger(ServiciosRestful.class.getName()).log(Level.SEVERE, "ERROR: Error al leer la respuesta.", e);
175             }
176 
177         }catch(MalformedURLException e){
178             Logger.getLogger(ServiciosRestful.class.getName()).log(Level.SEVERE, "ERROR: URL con formato incorrecto.", e);
179         }catch(ProtocolException e){
180             Logger.getLogger(ServiciosRestful.class.getName()).log(Level.SEVERE, "ERROR: Error durante la llamada HTTP.", e);
181         }catch(IOException e){
182             Logger.getLogger(ServiciosRestful.class.getName()).log(Level.SEVERE, "ERROR: Error de entrada/salida.", e);
183         }catch(JSONException e){
184             Logger.getLogger(ServiciosRestful.class.getName()).log(Level.SEVERE, "ERROR: JSON con formato incorrecto.", e);
185         }
186 
187         return result;
188     }
```

Figura 25. Procedimiento 'llamadaServicio'.

Ciertas funciones asociadas a servicios devuelven resultados del tipo ‘EstadoSistema’. Esta clase representa el estado tanto del sistema automático como del actuador, teniendo 3 posibles estados: desconocido, encendido/abierto y apagado/cerrado. La función 'interpretarEstadoSistema' toma el estado representado en formato JSON y devuelve un objeto 'EstadoSistema' correspondiente a dichos valores.

4.4.3 Registro de muestras

Como se ha señalado, la toma y registro de muestras es una parte fundamental de la aplicación,por lo que se va a mostrar cómo se lleva a cabo este proceso.

```
32      @Override
33      public void run() {
34          System.out.println("COMIENZO");
35          try {
36              valor = ServiciosRestful.getMuestraCO2();
37              Muestra m = Utilidades.crearNuevaMuestra(Constantes.ID_NIVEL_CO2, new Date(), new BigDecimal(valor));
38              mC.merge(m);
39          } catch (Exception ex) {
40              Logger.getLogger(MuestraTask.class.getName()).log(Level.SEVERE, null, ex);
41          }
42          System.out.println("Valor Actual: "+valor+" .");
43      }
```

Figura 26. Procedimiento de toma de muestras realizado periódicamente.

La toma de muestras es una tarea periódica, es decir, que va a realizarse a lo largo del tiempo automáticamente y en segundo plano. Para ello, vamos a usar una clase que ya

usamos para las tareas periódicas del servicio web, la clase 'TimerTask'. En este caso, al igual que en el servicio, la clase '*MuestraTask*' hereda de '*TimerTask*' e implementa su función en el método '*run*'. El funcionamiento es simple, se llama al servicio y se guarda la muestra con una marca de tiempo en la base de datos.

Para esta tarea, es posible alterar el periodo de la petición, aunque esto solo se puede hacer desde el panel de administración. El cambio de periodo conlleva una reprogramación de la tarea. Esto se hace estableciendo un nuevo '*Timer*' con el método '*Timer.schedule*'.

```
120                 segundosEspera = nuevoIntervaloEnSegundos;
121                 tiempoEspera = segundosEspera * 1000;
122                 accionTimer.cancel();
123                 accionTimer = this.nuevaTareaWS(String.valueOf(tiempoEspera));
124                 timer.schedule(accionTimer, 5000, tiempoEspera);
```

Figura 27. Reprogramación del 'Timer' con un nuevo intervalo.

4.4.4 Panel de control

En el panel de control podemos encontrar las funciones accesibles para alterar el funcionamiento del sistema. Existen 3 funciones básicas: control del sistema automático, control del actuador y control del nivel máximo de CO2.

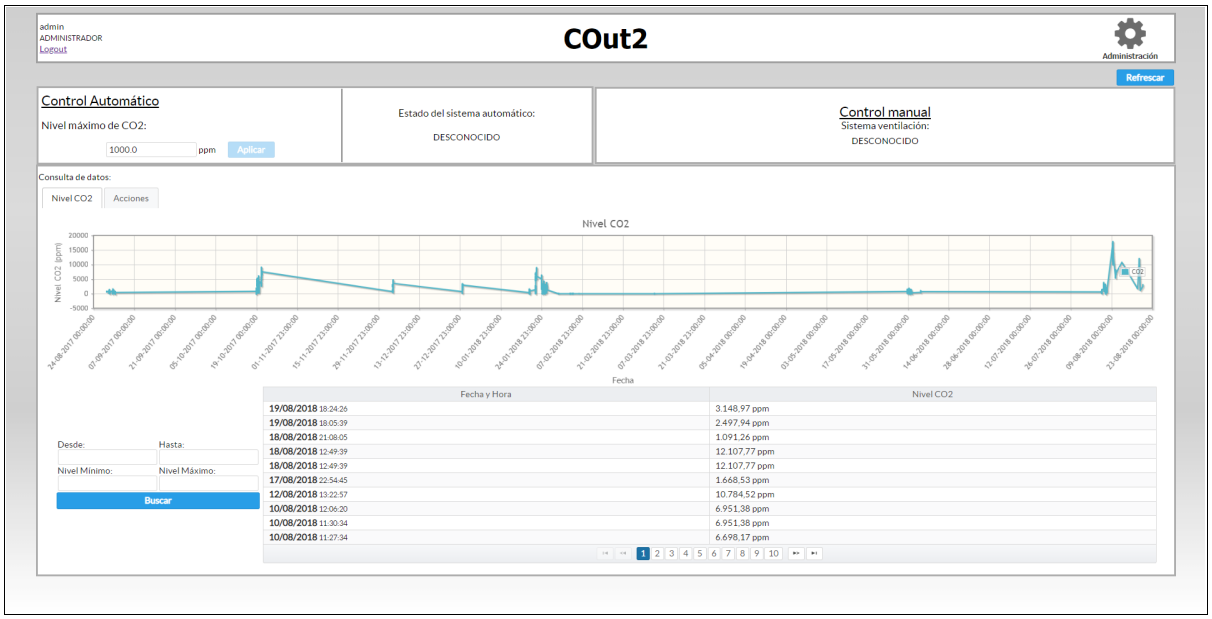

Figura 28. Interfaz gráfica del panel de control. Al no estar conectado el servicio y no obtener datos del estado, el sistema muestra los estados desconocidos y no permite alterar el nivel de máximo de CO_2.

Estos controles solo son visibles para ciertos usuarios, dependiendo del perfil que se posea. Cuando una acción se realiza, el panel se bloquea durante la ejecución de esta, evitando así que se realice más de una al mismo tiempo. También tenemos la opción de recargar el estado para obtener el estado actual del sistema, y así como el estado, el nivel actual establecido como máximo. Ante un error o por falta de respuesta, el estado puede

mostrarse como desconocido, bloqueando las opciones de acción sobre el sistema hasta obtener una respuesta satisfactoria de este.

En esta misma página tenemos un gráfico que representa las distintas muestras obtenidas a lo largo del tiempo. Podemos filtrar según algunos parámetros para así obtener una vista más detallada de una franja de tiempo más concreta.

4.4.5 Gestión de usuarios

Para los usuarios administradores además existe la posibilidad de gestionar los usuarios del sistema. La gestión de usuarios consiste en la creación de nuevos usuarios, el bloqueo y desbloqueo de estos, la edición de credenciales y el perfil de todos los usuarios.

Al inicio de la aplicación existe siempre un usuario administrador predeterminado, dando así la posibilidad de crear el restos a partir de este. Al crear un usuario, se le asigna una identificación y una contraseña. Esta contraseña no se guarda en base de datos, si no que se usa un sistema de encriptado (Scrypt de Lambdaworks. Anexo d), un algoritmo de cifrado especialmente pensado para contraseñas, ya que realiza un proceso pesado (en comparación con otros algoritmos) para evitar ataques de fuerza bruta.

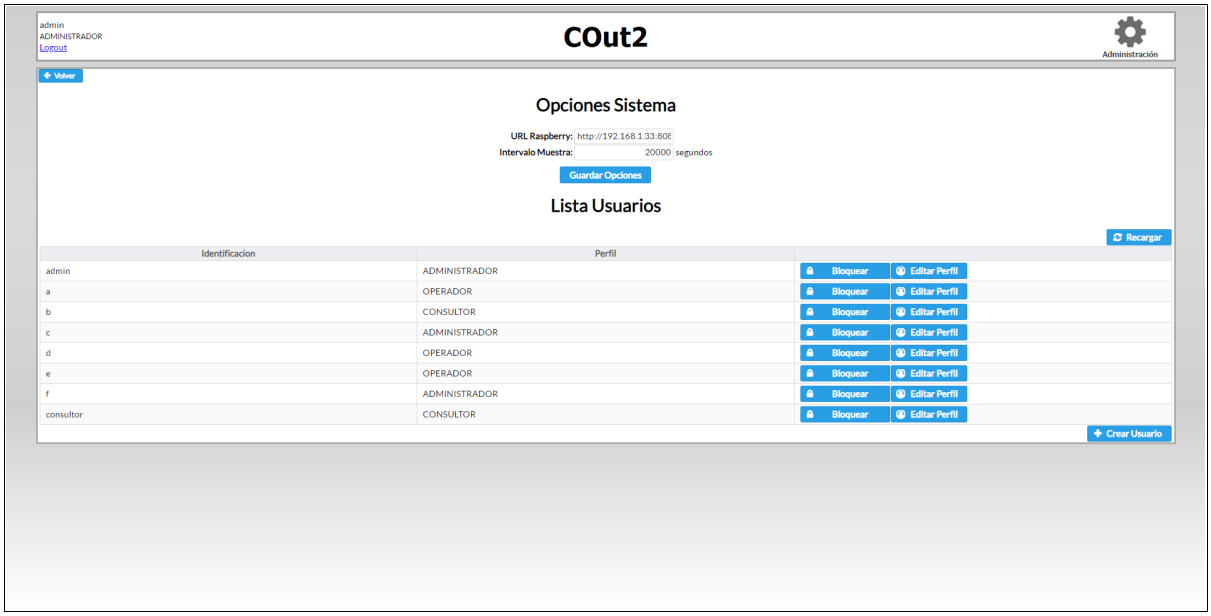

Figura 29. Interfaz de usuario del panel de administración, incluyendo tanto la gestión de la configuración como la gestión de usuario.

Para evitar accesos a usuarios no identificados, se ha establecido un filtro que, para toda página que no sea '/login', comprueba que el usuario esté logueado y si no lo redirecciona a '/login'. Así mismo se ha hecho para el caso contrario, en el que sí se está *logeado*, '/login' te redirecciona a '/panelPrincipal'.

Para controlar qué usuarios acceden al panel de configuración, existe un tercer filtro que comprueba el perfil del usuario ante el acceso a este. En caso de no ser administrador, es redireccionado a '/panelPrincipal'.

```
34      @Override
35      public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException {
36          SessionBean loginBean = (SessionBean)((HttpServletRequest)request).getSession().getAttribute("sessionBean");
37
38          if(!((HttpServletResponse)response).isCommitted()){
39              if (loginBean != null && loginBean.comprobarLogin()) {
40                  String contextPath = ((HttpServletRequest)request).getContextPath();
41                  ((HttpServletResponse)response).sendRedirect(contextPath + "/panelControl/panelPrincipal.xhtml");
42              }
43          }
44
45          chain.doFilter(request, response);
46      }
```

Figura 30. Filtro que comprueba el estado de identificación.

```
28      <filter>
29          <filter-name>FiltroLoggeado</filter-name>
30          <filter-class>filtro.FiltroLoggeado</filter-class>
31      </filter>
32      <filter-mapping>
33          <filter-name>FiltroLoggeado</filter-name>
34          <url-pattern>/panelControl/*</url-pattern>
35          <url-pattern>/facelets/*</url-pattern>
36      </filter-mapping>
```

Figura 31. Registro del filtro en 'web.xml'. Está asociado a cualquier página que no se la de login.

4.4.6 Registro de acciones

Por último, los usuarios registrados tienen acceso a un informe de acciones realizadas por usuarios en el '/panelPrincipal'. Dicho registro graba acciones como alteraciones en los usuarios o acciones con el sistema, así como una marca de tiempo, pudiendo ser consultadas en cualquier momento esclarecer posibles errores.

5 Pruebas

Finalmente, tras todo el desarrollo pasamos a la fase de pruebas. Durante el desarrollo se han ido probando pequeñas funcionalidades conforme se completaban para poder seguir con la siguiente, pero esta es la primera prueba de funcionamiento general en la que todas las partes toman partido.

En un primer momento, las lecturas no parecían ser muy estables y para nada se correspondían con los valores esperados. Tras investigar y revisar el datasheet del sensor se realizó en un error en la interpretación de la información.

En el *datasheet* podríamos ver una figura referente a un circuito, que inicialmente se interpretó como el circuito interno del sensor, y si bien la interpretación no era del todo errónea no se había reparado en un detalle. En dicho diagrama se puede ver la 'resistencia de carga' o 'Rl', utilizada para calcular el 'Rs' y por ende para calcular el nivel de CO^2. Ahí (en el *datasheet*) se habla de resistencia variable, por lo que se entendió que era algo aleatorio y que su valor era algo ya establecido, por lo que leyendo, al sugerir una 'Rl' de 20000 Ω se interpretó como el valor que usar en la función sin más necesidad de modificación. Sin embargo, esto no es así, ya que como luego se descubrió (está plasmado en el punto 4.2) es una resistencia que nosotros debíamos establecer.

Fijándonos en el módulo del sensor (como se puede ver en este[8] blog) ya trae incorporada una resistencia de 1000 Ω, pero el *datasheet* establece una recomendación de 20000Ω (entre 10000 Ω y 42000 Ω) por lo que se han añadido 2 resistencias en serie de 10000Ω .

Además, el primer ajuste realizado daba resultados muy inestables debido a que, como se indicaba en el *datasheet*, el sensor tiene un tiempo de calentamiento de entre 24 y 48 horas, tiempo que no se había esperado para dicho ajuste. Por lo tanto, tras solucionar los problemas con el circuito y con la fórmula de cálculo de 'Rs', se dejó conectado el sensor por un periodo aproximado de 24 horas y tras este tiempo, se volvió a ajustar el sensor con el aire como referencia, obteniendo esta vez un 'R0' de 250000 Ω frente a los 200000 Ω de la primera medición.

Una vez hecho esto, se ha preparado una caja de plástico a modo de habitáculo para introducir el sensor y ventilador y simular un espacio cerrado. Con todo montado la conexión se realiza totalmente de acuerdo a lo esperado, recibiendo el servicio todas las llamadas así como ejecutando los comandos Arduino. Las mediciones ahora son mucho más coherentes

(recordemos que no disponemos de un sensor profesional para comparar pero sí de los datos expuestos) y tienen mayor correlación con los datos esperados según la información encontrada. El sensor es extremadamente sensible a los movimientos, golpes, zarandeos,... por lo que cualquier temblor o traslado puede causar lecturas erróneas hasta el punto de estar un largo periodo de tiempo tomando lecturas totalmente disparatadas, pero esto es a causa del sensor en sí mismo, por lo que no es algo que abarcar ni que podamos solucionar, cosa que no importa demasiado dado a la finalidad de este tipo de sistemas. También, en contra del sensor, decir que no es muy preciso ya que las lecturas están discretizadas entre 0 y 1023, por lo que los saltos entre dos mediciones son amplios.

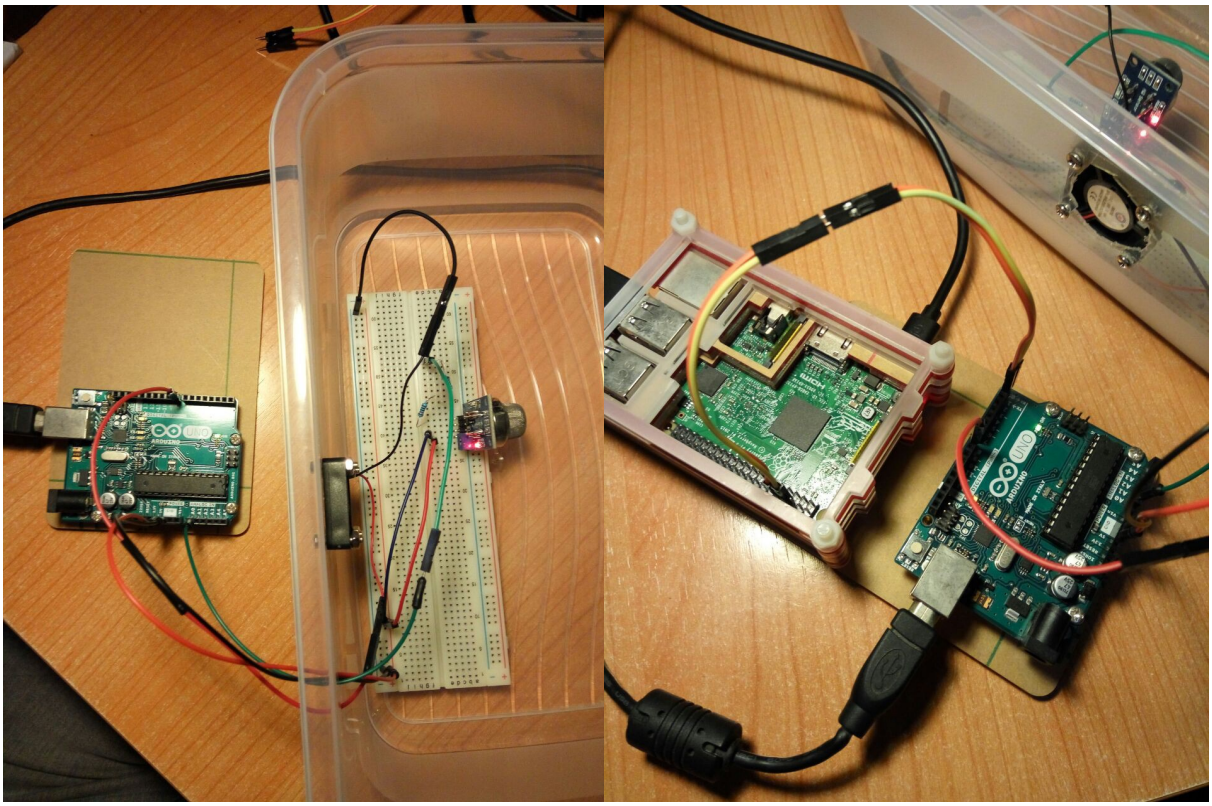

Figura 32. Montaje del sistema entre Arduino y Raspberry con parte del sistema de ventilación.

Con respecto al sistema en general la comunicación desde la web hasta Arduino es muy rápida pero la respuesta se hace esperar un poco, cosa que no es para nada crucial y con la que el sistema se puede considerar totalmente funcional y acorde a las características esperadas.

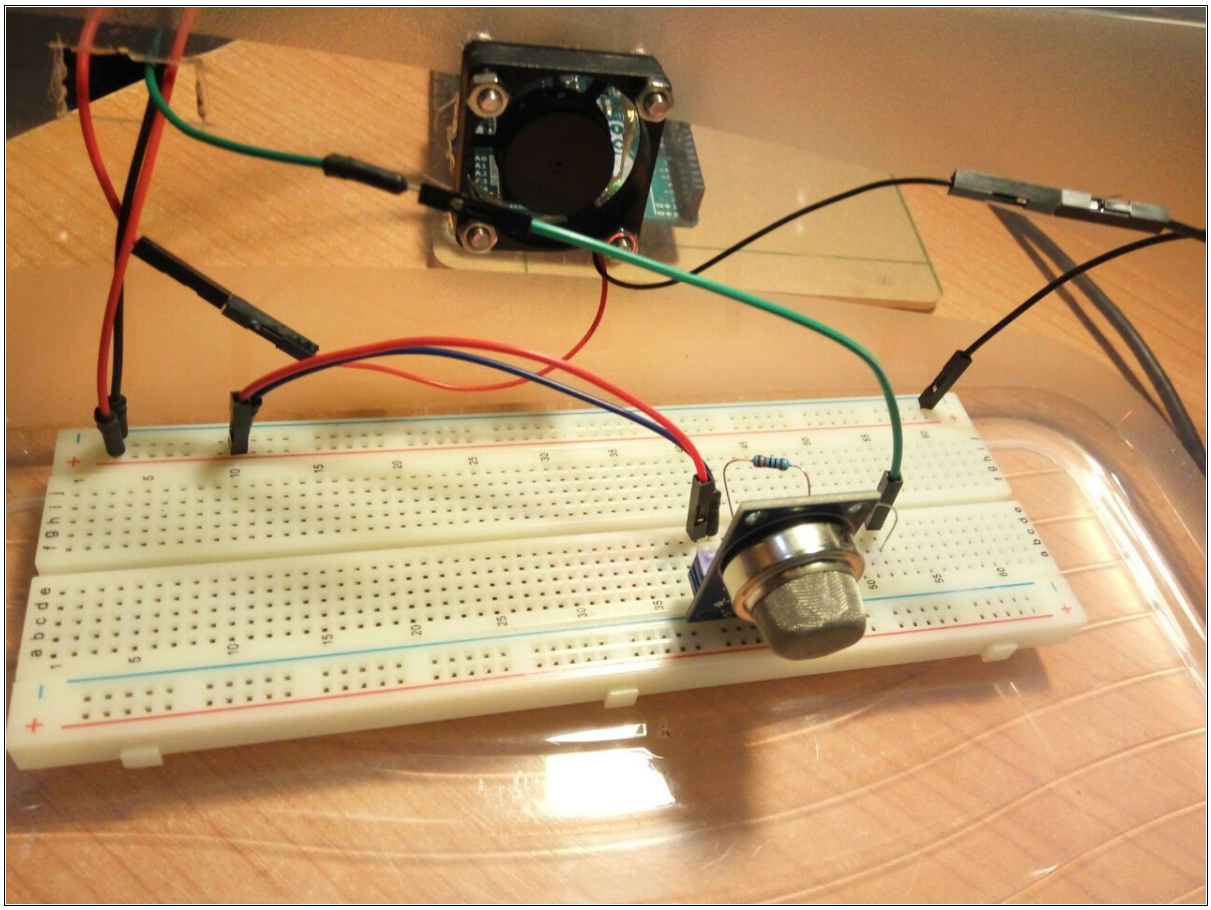

Figura 33. Montaje del circuito con el sensor y el actuador (ventilador). El cable rojo del ventilador acaba en el pin digital 8 de Arduino.

Por lo tanto y a la vista de los resultados, el proyecto cumple sus especificaciones conforme a lo establecido durante toda esta memoria y se da por concluido su desarrollo.

6 Futuras líneas

Debido al número de módulos del proyecto, existen varias líneas por las que ampliar la funcionalidad de este, siendo imposible haberlas realizado en este proyecto por estar enmarcado en un espacio de tiempo limitado. Por ello estas son algunas de las líneas por las que podría ampliarse este proyecto.

6.1 Implementación control de acceso en servicio web

Actualmente, el servicio web carece de cualquier tipo de seguridad ni de control en lo que respecta al remitente de las peticiones, por lo que cualquiera conocedor de algunos de los servicios o capaz de interponerse en las comunicaciones y leer el tráfico, es capaz de alterar el sistema sin impedimento alguno.

Por ello, existen diferentes protocolos basados en *tokens* que permiten tener un control sobre quién hace uso del servicio, evitando así posibles ataques que perjudiquen el correcto funcionamiento del sistema.

6.2 Implementación de nueva aplicación de control

6.2.1 Aplicación móvil

Al igual que se ha desarrollado una aplicación web capaz de acceder al control del sistema de una manera sencilla, con la posibilidad de hacerlo desde el móvil sería mucho más cómodo, por lo que haciendo uso de el resto del sistema actual, una implementación de un nuevo módulo de aplicación en forma de aplicación móvil podría aumentar la usabilidad y las posibilidades del sistema.

6.3 Ampliación de sistema de sensores y actuadores

Por último, el sistema de automático está basado en un solo sensor y actuador, pero los niveles captados en un gran habitáculo no son uniformes y pueden variar, por lo que una posible línea podría estar basada en aumentar la eficacia y precisión del sistema mejorándolo con una red de sensores que monitoricen aún mejor los niveles, siendo capaces incluso de ventilar sólo ciertas zonas y mejorando en mayor medida la calidad del aire.

7 Conclusiones

Este proyecto demuestra lo accesible que la tecnología se ha vuelto en estos últimos años, tanto económica como técnicamente. Esto es debido a la emergencia de propuestas como Arduino o Raspberry, tecnología barata, versátil y pensada para ser usada por usuarios con un nivel mínimo de conocimientos informáticos y electrónicos. Gracias a esto, se pueden ver disponibles numerosos proyectos basados en estos dispositivos, proyectos variados y abiertos para ser probados y aprender de ellos por parte de una comunidad dispuesta a resolver dudas y ayudar a los demás en general.

Esto se da en el mundo del desarrollo software en general, donde existen numerosas páginas donde desarrolladores ayudan a desarrolladores de manera desinteresada y abierta y, un gran catálogo de librerías y funcionalidades puestas a disposición del público general de manera también desinteresada y abierta como lo son por ejemplo *Scrypt* y *JSSC*, dos librerías usadas en este proyecto.

Por último, resaltar la intención del desarrollo de este proyecto en el que no sólo se pretendía llevar a cabo una cierta funcionalidad, sino que se han tenido siempre en mente dos ideas que aumenten el valor del proyecto como simple implementación.

La primera de estas es demostrar las posibilidades que puede ofrecer la cooperación de varios sistemas en el que cada uno se encarga de una tarea, de manera que se aumente la eficiencia buscando aprovechar los puntos fuertes de cada uno a la hora de asignar responsabilidades.

Por otro lado, crear un proyecto con módulos bien definidos fácilmente ampliables o reemplazables que ofrezcan a otros desarrolladores y futuros alumnos una base sobre la que construir sus propias variaciones. Esto es visible en distintos sitios como por ejemplo la base de datos, donde la tabla 'Tipo_Muestra' da una versatilidad enorme a la hora de añadir nuevos niveles que controlar, por ejemplo, y en la propia estructura del proyecto.

El uso de Arduino en este proyecto responde en gran parte a una suma de estos dos puntos, ya que Raspberry ofrece funcionalidad suficiente para llevar a cabo su función como servicio web y la de Arduino. Respondiendo al primer punto, Arduino es usado en sistemas de tiempo real, donde hace de su sencillez una virtud al permitir programas simples con respuestas rápidas y separando su tarea de la del servicio web evitamos posibles retrasos en las medidas y el control inducidos por fallos o procesos lentos como lo es la ejecución de un

servidor de aplicaciones. Además, separando estas dos funcionalidades en dos servicios diferentes hace posible alterar el sistema como se ha comentado en el apartado 'Futuras líneas' cambiando el *sketch* y el circuito sin modificar nada del servicio o creando un nuevo cliente independiente de la aplicación anterior.

Como conclusión, comentar la importancia de haber seguido una metodología adecuadamente, ya que ha sido fundamental para que la realización tanto del proyecto como de la memoria se hayan mantenido coherentes y ordenadas, especialmente para un proyecto de esta dimensión con diferentes módulos y tecnologías tan variadas.

8 Referencias bibliográficas

1. *MONTAÑO ARIAS, Noé M.; SANDOVAL PÉREZ, Ana L. Contaminación atmosférica y salud. 2007.* https://elementos.buap.mx/num65/htm/29.htm
2. *COMA, Jaume; BONET, Jordi; COMPANYS, Grupo Vall. Producción ganadera y contaminación ambiental. XX Curso de Especialización FEDNA: Avances en nutrición y alimentación animal. Fira de Barcelona, España, 2004, p.237-272.* http://www.produccionbovina.com.ar/sustentabilidad/46-ganaderia_y_contaminacion.pdf
3. *ROSALES-CASTILLO, José Alberto, et al. Los efectos agudos de la contaminación del aire en la salud de la población: evidencias de estudios epidemiológicos. Salud pública de México, 2001, vol. 43, p. 544-555.* https://www.scielosp.org/scielo.php?pid=S0036-36342001000600005&script=sci_arttext&tlng=pt
4. SOLÁ, Xavier Guardino. *Calidad del aire interior*. Instituto Nacional de Seguridad e Higiene en el Trabajo (INSHT), 2012. https://cdn.website-editor.net/f834eb0cae4d421a9adcc33432ab239d/files/uploaded/Estudio%2520CSIC%2520de%2520la%2520calidad%2520del%2520Aire%2520en%2520Espan%25CC%2583a%25202017.pdf
5. Jaycon Systems - https://www.jayconsystems.com/tutorials/gas-sensor-tutorial/
6. Singleton Pattern - https://sourcemaking.com/design_patterns/singleton
7. MathWorks “Serial Configuration” - https://www.mathworks.com/help/instrument/serialconfiguration.html
8. *Measure co2 with MQ-135 and Arduino uno, Rob - https://blog.robberg.net/mq-135-arduino/*

9 Anexos

a) MQ-135 Datasheet - https://www.olimex.com/Products/Components/Sensors/SNS-MQ135/resources/SNS-MQ135.pdf

b) JSON-Simple de fangyidong - https://github.com/fangyidong/json-simple

c) Java Simple Serial Connector - https://code.google.com/archive/p/java-simple-serial-connector/

d) Scrypt - https://github.com/wg/scrypt