Universidad de Valladolid

Escuela Técnica Superior de Ingenieros de Telecomunicación

Trabajo de fin de grado

Grado en Ingeniería de Tecnologías Específicas de Telecomunicación
Mención en telemática

Diseño y desarrollo de un sistema de monitorización y control del riego en una explotación agrícola

Autor
D. Luis Carlos Parra Rebollo

Tutores
Dr. D. Manuel Rodríguez Cayetano
Dra. Dña. María Jesús González Morales

Título	**Diseño y desarrollo de un sistema de monitorización y control del riego en una explotación agrícola**
Autor	**D. Luis Carlos Parra Rebollo**
Tutores	**Dr. D. Manuel Rodríguez Cayetano** **Dra. Dña. María Jesús González Morales**
Departamento	**Teoría de la Señal y Comunicaciones e Ingeniería Telemática**

Tribunal

Presidente	Pedro Chamorro Posada
Vocal	Manuel Rodríguez Cayetano
Secretario	Mª Jesús González Morales
Fecha	Noviembre de 2020
Calificación	

Dedicatoria

A Dios, porque sin él nada es posible.
A Dani, por los 21 años que nos has regalado, y por no poder escribir esto sin lágrimas.
A Manuel y a Chus, por la dedicación prácticamente ilimitada que han tenido conmigo.
A mis padres, porque se lo debo todo.
A LP y a Carmen, por apoyarme siempre.
A los amigos que he descubierto en este trayecto.
A Jesús y a Tomás, porque me hubiera gustado que me vierais cerrar esta etapa.
A mi abuela y mis hermanos.

Resumen

La automatización sigue una tendencia de crecimiento en todos los sectores de la actividad productiva. Como ejemplo de esta tendencia, podemos observar la evolución de la automatización en la industria, la logística y la agricultura. En la agricultura, los sistemas de control de riego programados contribuyen a mejorar la eficiencia del sistema de riego.

Este trabajo ha desarrollado un sistema de monitorización y control de riego en una explotación agrícola, el cuál se ha presentado como solución a un problema de accesibilidad al sistema existente en la explotación. Concretamente, el sistema previo utilizaba la tecnología bluetooth, lo que impedía controlar las válvulas que permiten la aplicación del agua sobre el terreno, ubicándose a más distancia de 10 metros de la válvula. El sistema del presente trabajo permite el acceso desde dentro y fuera de la explotación (mediante la red WiFi interna o mediante Internet), a cualquier distancia de la válvula a controlar.

Para la implementación del sistema se han utilizado diversas tecnologías. Java se ha empleado como el lenguaje de programación de la aplicación web de control, con Hibernate y DerbyDB para la gestión de las bases de datos. Restlet para implementar un servidor web restful con uso del protocolo seguro HTTPS. Además, la implementación del sistema de autenticación de usuarios también se ha apoyado en el framework Restlet. Y Freemarker y Bootstrap para implementar la interfaz de usuario. Por otro lado, en el diseño de la aplicación mencionada se ha utilizado la arquitectura propuesta en ROA/D. También hemos utilizado placas de desarrollo Arduino (el modelo MKR WiFi 1010), programándolas con su propio lenguaje y utilizando aWOT para implementar un servidor web restful en la placa.

Se ha desarrollado utilizando una metodología de desarrollo similar a la del «modelo en cascada». Dividiendo nuestras fases en 5 (aunque las dos fases previas a la última tengan cierto solapamiento temporal): análisis, diseño, implementación, pruebas y despliegue. Como parte de las pruebas, el sistema se ha probado, con resultado satisfactorio, en un entorno real en una explotación agrícola de la Escuela de Ingeniería Agrícola INEA.

Palabras Clave

Agroindustria 4.0, automatización de riego, Arduino, Java, Hibernate, Restlet, servicios web restful, REST, ROA.

Abstract

About Automation, an upward growth is being observed looking at all the economic sectors. As an example of this trend, evolution of automation in industry, logistics and agriculture can be observed. About agriculture, scheduled irrigation control systems help to get better the efficiency of the irrigation system.

An irrigation monitoring and control system in a farm has been developed through this paper. It has been proposed as a solution to an accessibility issue of the farm current system. Specifically, the issue was about the impossibility of remotely controlling of the irrigation valves (It was just possible to control them from less than 10 meters away due to the use of bluetooth technology) that allow the water to reach the ground. Developed system allows the access from inside and outside the farm (through the internal WiFi network or Internet), to any distance of the irrigation valve.

Several technologies have been used to implement the system. Java has been used as the programming language of the control web application, with Hibernate and DerbyDB for database management. Restlet has been used to implement a restful web server with authentication and use of HTTPS. And Freemarker and Bootstrap to implement the user interface. Furthermore, in the design of the mentioned application the proposed architecture in ROA/D has been used. We have also used Arduino development boards (MKR WiFi 1010 model), programmed them with the Arduino language and used aWOT to implement a restful web server on the board.

The system has been developed using a development methodology similar to the «Waterfall Model». We have split the process into 5 phases: analysis, design, development, testing and deployment. As part of the testing, the developed system has been tested, successfully, in a real environment on a farm of the Agricultural Engineering School INEA.

Keywords

Agribusiness 4.0, irrigation automation, Arduino, Java, Hibernate, Restlet, restful web services, REST, ROA.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1. Motivación

Hoy en día, la automatización es una tendencia clara en los principales sectores tradicionales de la actividad económica: en la primario secundario y terciario. En los tres, encontramos áreas automatizados parcial o totalmente (parcialmente es referido a que parte del proceso productivo este automatizado y otra parte no). Por poner algunos ejemplos, podemos observar un fuerte de crecimiento en la industria (parte del sector secundario): según [KH16], entre los años 2010 y 2014 el número de robots industriales vendidos ha crecido, aproximadamente, un 17 % cada año (de media entre los años del periodo citado) y, aún con algunos años de estancamiento debido a la crisis ecónomica de 2008 [KH16], una fuerte tendencia de crecimiento es observable en el periodo 2002-2014 [KH16]. Por su parte, en los servicios (parte del sector terciario) también se marca esta tendencia al automatizarse parte de las tareas como aquellas relativas a la logística (automatizadas con vehículos de guiado automáticos ó camiones autónomos por ejemplo), siendo la tendencia en logística más acusada que la comentada anteriormente, con un crecimiento de los robots producidos para este área superior al 80 % entre los años 2010 y 2014 [Kar+15] y, de aproximadamente el 37 % en el sector servicios en general [Kar+15].

Según [Mah+20], la automatización también es una tendencia en el sector de la agricultura (parte del sector primario). Esto se debe a que permiten mejorar el control sobre los cultivos y la eficiencia [Mah+20]. Concretamente, los sistemas de riego automático programado contribuyen a esta mejora de la eficiencia permitiendo «la aplicación del agua en el momento más adecuado para la planta» [Lio04] o cualquier otro tipo de cultivo.

Por todo ello, es sumamente interesante realizar un trabajo sobre el control y monitorización del riego en el que se trabajen con tecnologías útiles para otras posibles automatizaciones en explotaciones agrícolas. En la explotación objeto del la implementación de este proyecto existía previamente otro sistema de control del riego programado basado en la tecnología bluetooth, este sistema era susceptible de varias mejoras que, según referían sus usuarios actuales, eran: permitir el control del sistema a cualquier distancia del punto de aplicación del agua, permitir el control desde fuera de la explotación y facilitar la instalación de nuevos puntos de aplicación del agua.

1.2. Objetivos

El objetivo del presente trabajo es diseñar un sistema que permita la monitorización y control remotos de un sistema de riego, de forma que los encargados del mantenimiento de un cultivo o terreno puedan controlar el riego de una forma sencilla, eficiente y cómoda.

El sistema debe ser accesible desde cualquier parte (incluso fuera del recinto o cerco en el que se encuentre), para ello, es necesario que el sistema este disponible desde Internet. Resulta evidente la necesidad de implementar la interfaz de acceso al sistema como una aplicación web.

También debe permitir el control de las válvulas (mediante un circuito de control electrónico) que controlen el paso final del agua hacia el terreno. Y, con ello permitir controlar si el terreno debe ser regado en ese momento (abrir la válvula) o no (cerrar la válvula).

No solo se debe permitir el control de las válvulas, sino su monitorización, es decir, se debe permitir conocer el estado de la válvula (abierto o cerrado), dado que sería altamente ineficiente tener que acudir al terreno para comprobar si está siendo regado o no, cuando lo que buscamos es dar un control independiente del lugar donde este ubicado el usuario en un momento determinado.

Asimismo es necesario que el sistema permita al usuario configurar temporizadores con dos instantes temporales (que el usuario seleccionará indicando la hora y el día de la semana), uno que implique la apertura de la válvula y otro que implique el cierre de la válvula. Esto permite que un control programado y automático del riego.

Una vez iniciado el sistema, las válvulas deben poder añadirse al sistema de una forma sencilla que permita que los usuarios del sistema no dediquen tiempo y esfuerzo al aumento del terreno en el cual está implantado este sistema de control de riego.

1.3. Metodología

Para el desarrollo del sistema se ha utilizado una metodología basada en el modelo «en cascada» similar al definido en [Def88].

Las fases que vamos a ejecutar se basan en la parte del documento relativa al software y son las siguientes:

- Análisis de requisitos: En esta fase definimos el conjunto completo de requisitos que el sistema software ha de cumplir. La definición de estos requisitos ha de hacerse de una forma uniforme para todos los requisitos del conjunto (de la misma forma para todos).

- Diseño: El diseño va a constar de dos partes. Una es el diseño de la arquitectura, en el cual se deciden las partes de las que se va a componer el sistema software y en que van a consistir cada una de esas partes. Por otro lado, el diseño de los casos de uso en el que se define la interacción necesaria entre los diferentes componentes software y los diferentes escenarios para completar el caso de uso. En la definición de estas interacciones se debe asociar cada caso de uso a los requisitos (de entre los previamente definidos) que cumpla.

- Implementación: Esta fase se intercala temporalmente con ciertas partes de la fase de pruebas. Consiste en desarrollar los componentes software, servicios y casos de uso acorde a lo especificado en la fase de diseño.

- Pruebas: Se van a realizar tres tipos de pruebas, en función de la base (el objeto de las pruebas) y del alcance:

  - Pruebas unitarias: son aquellas en las que se prueban los componentes software de forma aislada (sin interacción con el resto de componentes). Se realizan después de la implementación de cada componente.
  - Pruebas de integración, en las que vamos a probar los diferentes casos de uso aislando cada subsistema, de forma que se identifique con claridad la fuente del error. Se dividirán en conjuntos de pruebas según el caso de uso que se vaya comprobar. Estos conjuntos de pruebas se ejecutan al finalizar el desarrollo completo de cada caso de uso.
  - Pruebas de aceptación: son aquellas en las que el objeto de la prueba es el sistema software al completo (con los diferentes subsistemas en comunicación) y, de la misma forma que en las pruebas de integración, la forma de separar los diferentes conjuntos va a ser basándonos en los diferentes casos de uso. Se deben ejecutar todos los conjuntos de pruebas de aceptación antes de realizar el despliegue.

- Despliegue: Puesta en marcha del sistema completo.

Se puede observar que en nuestro proceso, a diferencia de el detallado en [Def88], no dividimos el diseño en preliminar y detallado, esto se debe a la diferencia de la complejidad del proyecto al que están orientados ambos procesos (el aquí expuesto y el detallado en [Def88]).

1.4. Estructura del documento

El trabajo se compone de 5 capítulos más el que se está exponiendo (introducción al resto del documento). El segundo capitulo es un estudio de las tecnologías utilizadas para el desarrollo del sistema. En el tercer capitulo se exponen en detalle la primera fase del proceso de la metodología de desarrollo ya citada en la sección 1.3. Por su parte, el cuarto capítulo detalla la fase de diseño del proceso y el quinto realiza algunas aclaraciones sobre la implementación (tercera fase) y expone con detalle el plan de pruebas seguido (cuarta fase). Y, finalmente, se desarrollan las conclusiones y las lineas futuras de trabajo, que son los caminos en los que pensamos que deben continuar los avances relativos al presente trabajo.

Capítulo 2

Tecnologías relacionadas

2.1. Tecnologías de desarrollo software

2.1.1. REST

REST (del inglés *Representational State Transfer*) es un conjunto de restricciones para arquitecturas de red que proviene de varios estilos [Fie00], abstrae la información en forma «**recursos**» que es transmitida a través de **representaciones**, es definido para ayudar a implementar sistemas distribuidos y es independiente del protocolo utilizado [Dog18]. Las restricciones REST son: arquitectura cliente-servidor, sin estado, caché, interfaz uniforme y opcionalmente código bajo demanda [TB12].

- Arquitectura cliente-servidor: Esta restricción se refiere a la separación nítida entre el servicio y el cliente del servicio. De forma que, mientras el cliente se encarga de enviar las solicitudes que necesite, observar la respuesta y, actuar como considere según la respuesta recibida, el servidor se encarga de realizar las acciones solicitadas (en caso de que concuerden con el servicio ofrecido) antes de dar la respuesta correspondiente (incluyendo en ella descripciones de los errores producidos durante la realización de la tarea, en caso de que se hayan producido) [TB12].

- Sin estado: Las solicitudes deberán ser independientes del resto de solicitudes, es decir, toda la información que se necesite para entender la acción solicitada debe estar incluida en la solicitud [TB12] y no se deben solicitar acciones previas al cliente para responder a su solicitud.

- Caché: Las respuestas a las solicitudes deben especificar si esa respuesta se puede almacenar en la memoria caché o no. En los casos en los que se especifique afirmativamente, el cliente podrá utilizar esa respuesta para otras solicitudes [Fie00].

- El servicio se provee por medio de recursos, que como comentábamos en la definición de REST previa, es la forma en la que se abstrae la información en REST, pero no contiene información (de hecho, como se señala en [Fie00], pueden existir recursos que correspondan a «conjuntos vacíos», para permitir que se pueda crear posteriormente información con correspondencia a ese recurso). Estos recursos se deben manipular y transmitir por medio de representaciones, las cuales contienen el estado de cada conjunto de información correspondiente al recurso en un formato (por ejemplo, JSON[1], HTML, etc), denominados tipos de medio (del inglés *media-type*) que dependerá de cada implementación.

[1] *JavaScript Object Notation* (JSON) es «un formato de intercambio de datos». No depende del lenguaje utilizado y es «basado en texto»[Bra14]

- Interfaz uniforme: La interfaz de las arquitecturas que siguen este estilo deben proveer una misma forma de acceder a los recursos. Esta forma es mediante el uso de métodos estandarizados (por ejemplo, los métodos HTTP) [Fie00].

- La forma de señalar el destino de la petición es mediante la utilización de identificadores de recurso [Fie00]. Estos identificadores deben corresponderse de forma univoca con un recurso (no pueden existir varios recursos que compartan un mismo identificador). Además deben, según [Fie00] ajustarse bien «a la naturaleza del concepto identificado».

2.1.2. Arquitectura MVC y metodología ROA/D

El software se ha diseñado haciendo una clara separación de capas. Se diferencian claramente la vista, el controlador y el modelo, estos conforman los componentes de la arquitectura denominada de Modelo-Vista-Controlador (MVC) [Bur92][PP06], pero en este caso se añade otra capa, denominada de recursos, conformando la arquitectura propuesta en la metodología denominada «Resource-oriented Analysis and Design» (ROA/D)[LB13]. Esta diferenciación se hace con el objetivo entre otros, de conseguir facilitar la implementación de un servidor que cumpla los requisitos REST, un acoplamiento reducido y una escalabilidad elevada.

- La vista controla la parte gráfica de la aplicación [Bur92], lo que el usuario ve. Se comunica con la capa de recursos a través de algún tipo de comunicación que cumpla con las restricciones REST. [Bur92][LB13]

- La capa de recursos interpretan las entradas de los actores del sistema y de otros recursos, es decir, los datos que los actores introducen y las solicitudes de ejecución de acciones que realizan. Se comunica con la capa inmediatamente superior (vista) implementando un servidor que acepte peticiones HTTP (GET, PUT, POST...) y que cumpla con las restricciones REST, entre ellas destacamos: disponer de una interfaz uniforme (que será utilizada por la vista para solicitar operaciones sobre recursos), no mantener el estado de las peticiones, etc. [LB13]

- Los controladores realizan la gestión de partes o funcionalidades específicas y proveen una interfaz a la capa de recursos. De esta forma cada controlador se comunica con la capa de recursos y con el modelo, para ello tendrá instancias del modelo y será instanciado por los recursos que invocarán las rutinas que el controlador pone a su disposición. [Bur92]

- El modelo se encarga del control directo de los datos y el comportamiento del sistema[Bur92], es decir, modifica los datos directamente (cuando le es solicitado), devuelve los datos que se le soliciten... y ejecuta las acciones finales del sistema.

2.1.3. Servicios

Generalidades

En [Bar03] se define un servicio como un grupo de funciones definidas, autónomas y que no dependen del contexto o de otras funciones. Por el contexto del capitulo mencionado, podemos concluir que con la palabra función se refiere a una solución para una realidad (por ejemplo, la realidad podría ser una base de datos y la solución una de las formas de gestionarla). Donde dice definida, nos indica claramente que la forma de usarlo debe estar completamente descrita y documentada.

Servicios web

La diferencia que tiene un servicio web con otro tipo de servicios, basándonos en la definición previa de servicio y la de servicio web dada en [HB04], es utilización de estándares relacionados con la web (por ejemplo, WSDL, SOAP, HTTP, etc). Por lo tanto podríamos concluir que un servicio web es una función definida, autónoma, que no depende del contexto o de otras funciones y que utiliza estándares relacionados con la web (por ejemplo, WSDL, SOAP, HTTP, etc).

Servicio web restful

Un servicio web restful es aquel servicio web que cumple con las restricciones definidas para REST.

Al ser servicio web, una de sus características es la utilización del protocolo HTTP. Como se explica en el primer RFC (*Request for Comments*) sobre HTTP [BFF96], los identificadores de recurso utilizados son los URI (*Universal Resource Identifiers*) definidos para el sistema WWW (*World Wide Web*) en el RFC 1630 [Ber94]. Por lo tanto estos serán los identificadores utilizados en los servicios web restful para cumplir con el contrato uniforme definido como restricción del estilo REST.

Por la misma razón, los métodos de ese mismo contrato uniforme serán los definidos en el RFC 2068 [RGB97] que amplia los previamente citados en [BFF96]: OPTIONS, GET, HEAD, POST, PUT, PATCH, DELETE, LINK, UNLINK y TRACE.

2.1.4. El framework *Restlet*

Restlet es un framework[2] para el desarrollo en el lenguaje de programación java, que facilita el desarrollo de servicios web que cumplan con las restricciones de la arquitectura REST [Tal20c].

Concretamente, Restlet nos facilita el desarrollo tanto de servidores HTTP como de clientes. Aunque los servidores no necesitan de clientes que hayan sido desarrollados mediante el uso del framework sino que las solicitudes que realicen cumplan con las exigencias de un servidor HTTP.

Restlet proporciona clases para simplificar el desarrollo de las tareas de inicialización de la aplicación (clase Component [Tal20a]), la asignación de URLs a recursos (clase Router [Tal20d]) y la implementación de los propios recursos (que se basarán en la clase predefinida ServerResource [Tal20b]).

Además, proporcionará otras clases auxiliares para la gestión de la autenticación de usuarios y otras tareas comunes a las aplicaciones que implementan un servicio web de tipo restful. Todas estas clases auxiliares que utiliza directamente el desarrollador de la aplicación se apoyarán en otras clases internas encargadas de los aspectos más básicos de la comunicación con los clientes.

2.1.5. *Freemarker*

Freemarker es un motor de plantillas HTML para java. Un motor de plantillas es una API[3] que «combina una o más plantillas con un modelo de datos para producir uno o más documentos resultantes» [BA09].

[2] Un framework es un conjunto de clases, procedimientos, etc. que implementan «un diseño abstracto para una familia de problemas» [JF98].

[3] Una API (del inglés *Application Programming Interface*, en español «Interfaz de programación de aplicaciones») es una definición de la forma de interacción entre dos componentes software, que ayuda a los desarrolladores a construir una aplicación [De17].

En el caso de *freemarker*, sus plantillas se especifican en HTML con extensiones propias que permiten, entre otras facilidades, definir variables cuyos valores serán fijados dinámicamente desde la aplicación java. De esta forma, las plantillas contendrán código HTML estático y zonas que serán sustituidas por valores enviados desde la aplicación java en el momento en que se deba crear el HTML completo de respuesta a una petición del cliente. Este mecanismo permite evitar tener que generar el mismo código HTML común desde el servicio web programado en java en cada respuesta a una petición sobre el mismo recurso.

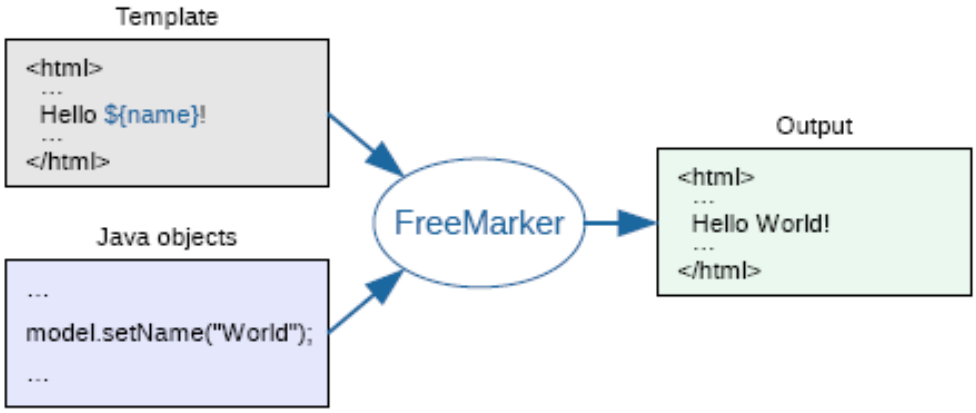

FIGURA 2.1: Flujo de funcionamiento de freemarker [extraído de [Apa20d]]

En la figura 2.1 se muestra un ejemplo sencillo del funcionamiento de la API. La plantilla (en inglés template) se muestra un código «HTML» en el que se incluye la cadena «${name}», de esa forma se indica que esa cadena tiene que ser sustituido por el valor de la variable «name» del modelo. En ese tipo de cadenas se puede incluir, no solo variables sino también ciertos métodos provistos por la librería de «Freemarker», por ejemplo para calcular el promedio entre dos números (a y b) se haría de así «${avg(a, b)}», a su vez esos números podrían ser sustituidos por variables incluyéndolos de la forma descrita previamente. Las variables mencionadas podrían ser objetos java, a cuyos atributos accederíamos en la plantilla como si todos ellos fueran públicos: «${objeto.atributo}»

En la parte de la figura 2.1 en la que se especifica el código java se hace referencia a un objeto «model», este objeto es del cual se extraerán después los valores de las variables invocadas desde la plantilla. Este objeto (siguiendo lo expuesto en [Apa20a]) debe ser un objeto de tipo clave-valor (como un «Map») o un objeto de tipo «javaBean» (objetos simples, con «getters» y «setters» y serializables). En la figura es uno de los últimos dado que se asigna el valor del atributo «name» mediante la utilización del método llamado «setName» (en lugar de «put» como correspondería en un «Map»).

Finalmente observamos como la quedaría el código resultante al asignar a la variable su valor.

2.1.6. Bootstrap

Bootstrap se puede definir como una serie de ficheros CSS[4] y, adicionalmente, ficheros en el lenguaje javascript , que sirve principalmente al desarrollador de «front-end»[Kra16] para

[4]Según el consorcio *World Wide Web* [Con20], CSS es el lenguaje para codificar la «presentación de páginas web». En los ficheros CSS (denominados hojas de estilo), se indican instrucciones de ese lenguaje para indicar las características estéticas (como el tamaño de la letra, el color del fondo, la posición en la página, etc) asociadas a elementos HTML o atributos HTML, tales como clases.

simplificar el desarrollo de páginas web con el objetivo de que tengan un aspecto agradable y sean fácilmente utilizables en todo tipo de dispositivos (ordenadores, tablets, teléfonos móviles, etc).

Además de lo comentado previamente, hay que tener en cuenta otras ventajas de la utilización de Bootstrap, como puedan ser:

- Amplia gama de navegadores soportados: Los ficheros HTML y CSS no son interpretados de la misma manera por todos los navegadores, ni en todas la versiones de un mismo navegador. Bootstrap, sin embargo, debido a la forma en la que está implementado, es interpretado de la misma manera por una amplia gama de navegadores [tea20b].
- Accesibilidad: La utilización de esta herramienta facilita el cumplimiento de las pautas marcadas en el estándar WCAG [tea20a], el cual indica ciertas formas de desarrollar tecnologías web para hacer esos contenidos accesibles a «personas con discapacidades» [Cal+08].
- Usabilidad: Normalmente el desarrollo de interfaces estéticamente amigables, intuitivas, etc. dilata enormemente el tiempo de desarrollo de una aplicación web. La utilización de Bootstrap cambia esto, dado que las tablas, cajas, formatos de los textos, etc. ya están desarrollados y solo se necesita seleccionarlos, incluirlos y personalizarlos mínimamente.

2.2. Bases de datos

2.2.1. Generalidades

Basándonos en [Teo+11] una base de datos es una colección de datos (que son la unidad de información «con nombre más pequeña con correspondencia en el mundo real») relacionados entre sí.

Según Codd [Cod90], los datos (de una base de datos) son de dos tipos: «atómicos y compuestos». Donde atómicos son aquellos que no se pueden descomponer en unidades más pequeñas y compuestos son aquellos que se pueden descomponer en atómicos.

Las bases de datos relacionales son aquellas que siguen el modelo relacional. Sus principales características son las siguientes:

- Denomina tupla a cada registro de la base de datos. Cada elemento de la tupla tiene un nombre (que será el nombre de la columna) y no solo un valor [Cod90].
- Todas las tuplas que representan datos con la misma estructura deben presentar las mismas columnas (y con los mismos nombres) [Cod90].
- No puede haber tuplas duplicadas (registros con los mismos datos en todos los campos) [Cod90]. Además el identificador debe ser único.
- Se deben identificar mediante el «contenido informativo» y no mediante identificadores añadidos a propósito como «identificadores de registro» [Cod90], siempre que sea posible. Por ejemplo, si la información que va a contener la tupla son los datos de una persona y, entre ellos, se incluye el DNI, este sería un buen identificador, dado que es parte de la información que se desea almacenar y es único.
- La «relación» es el único tipo de dato compuesto que existe en este modelo. [Cod90].

2.2.2. DerbyDB

DerbyDB (ó Apache Derby) es una herramienta de Apache destinada a crear y gestionar bases de datos relacionales utilizando java.

Es una de las múltiples soluciones basadas en JDBC. JDBC es una API, cuya función más destacada es la de permitir acceso a bases de datos desde el lenguaje de programación java y, además permite el acceso desde otras fuentes de datos como las «hojas de cálculo» [Ora20]. Entre las ventajas de la utilización de DerbyDB, se encuentran las siguientes:

- Es una herramienta de código abierto [Apa20c], lo que implica que uso es libre y gratuito.
- Permite que «el motor» (aquello que nos ofrece una interfaz para manejar el sistema gestor de la base de datos) esté «embebido» en la aplicación, esto quiere decir que ese motor de la base de datos se ejecute dentro de la propia aplicación y lo único que quede fuera sean los ficheros con los datos y la estructura [Apa20b]. Esto determina que no sea necesario instalar un servidor especifico para la base de datos.

2.2.3. Hibernate

Hibernate ORM es un framework que facilita el desarrollo de aplicaciones en java que gestionen bases de datos. Lo hace utilizando ORM (*Object-Relational Mapping* o correspondencia objeto-relación). ORM es un sistema que establece una forma de convertir una clase, de un lenguaje de programación orientada a objetos, en la estructura de una tabla de una base de datos relacional [Hib20]. Esta correspondencia, a su vez, permite tratar a cada registro de esa tabla como un objeto (del lenguaje de programación orientada a objetos).

Una de las grandes ventajas del uso de este tipo de frameworks es, directamente, el hecho de poder gestionar registros de la base de datos como objetos java, evitando la necesidad de utilizar el lenguaje SQL para consultar o modificar la información contenida en la base de datos.

2.3. Arduino

Las placas de desarrollo Arduino son circuitos programables capaces de leer el voltaje de un diverso número de entradas (que depende del modelo de la placa), y de modificar el voltaje de las salidas [Ard18]. Estas «lecturas» de los voltajes de entrada al circuito y variaciones de los voltajes de salida del circuito son controladas por el programa *software* ejecutado. Este programa se desarrolla en un lenguaje propio basado en C++ con un ordenador y, desde el mismo, se puede enviar a la placa para su ejecución de forma muy sencilla, gracias a un entorno de desarrollo de «*software* abierto» de la propia marca. La programación con Arduino se ve beneficiada por la gran «versatilidad» y el «soporte masivo» que tiene asociada debido al gran número de usuarios y empresas que participan desarrollando extensiones y librerías [WAM11].

2.3.1. La biblioteca aWOT (*Arduino web server library*)

AWOT es una biblioteca que permite construir servicios web de tipo restful para acceder a los recursos de una placa Arduino (fundamentalmente, para consultar los valores de puertos de entrada y enviar valores a los puertos de salida) [aWO20a]. Permite que la interacción con una placa Arduino se realice mediante el protocolo HTTP y siguiendo los criterios de REST, con las ventajas que esto implica (ver sección 2.1.1).

Para implementar un servicio web restful utilizando esta biblioteca es necesario dar un reducido número de pasos:

- Encaminamiento: Al definir el encaminamiento con este framework lo que estaremos haciendo es asociar cada URL, conjuntamente con una operación HTTP, a una función que se encargará de gestionar las operaciones internas que se deben desencadenar a raíz de la petición y, la respuesta que el servidor va a dar al cliente en función de los detalles de la petición. La forma de hacer este encaminamiento es sencilla, se hace invocando una serie de métodos sobre una instancia de la clase «Application», estos métodos serán «get(const char* urlPattern, Router::Middleware* middleware)» para la petición GET, post(const char* urlPattern, Router::Middleware* middleware) para la petición POST, etc.[aWO20b] Donde el parámetro «urlPattern» hace referencia al URL de la solicitud y «middleware» es un puntero que apunta a la función que gestionará el comportamiento del servidor ante la solicitud.

- Definir las funciones de respuesta: En estas funciones, comentadas en el punto previo, debemos definir dos parámetros; el primero de ellos es «Request &req», este parámetro nos permite acceder a las propiedades y detalles de la solicitud, el otro es «Response &res» que nos permite configurar y enviar la respuesta.

- Invocar el procesamiento de las solicitudes cuando exista un cliente conectado al servidor, a través de la llamada «Application::process». Un ejemplo de como comprobar si el cliente está conectado e invocar el método process está en la referencia de AWOT en el apartado «Application::process».[aWO20b]

2.4. Válvulas de control de riego

Una electroválvula es un elemento electromecánico que permite abrir o cerrar el paso de agua a través de un conducto mediante una señal eléctrica.

Las electroválvulas, según el tipo de alimentación eléctrica que requieran, las podemos clasificar en dos tipos: las de corriente alterna y las de corriente continua. Todas ellas funcionan gracias a la creación de un campo magnético al circular la corriente eléctrica por el solenoide (una de las piezas que componen la electroválvula). Este campo magnético se utiliza para cambiar de posición la pieza que determina si se permite o se bloquea el paso de agua a través de la electroválvula

Generalmente, para abrir o cerrar la electroválvula basta con generar pulsos eléctricos de un ancho de unas decenas de milisegundos (los máximos y los mínimos dependerán de la electroválvula concreta) y de una amplitud que varía según el solenoide que vaya acoplado en la electroválvula, aunque con un rango estándar para la mayor parte de las existentes en el mercado. Las amplitudes del pulso típicas para electroválvulas de corriente continua suele ser de 12 ó 9V y la duración típica del pulso oscila entre 15 y 100 ms, habiendo otras que exceden este rango de ancho de pulso holgadamente.

Capítulo 3

Análisis

3.1. Descripción del sistema

El sistema que se ha desarrollado en este proyecto se denomina *smartgranja*. *Smartgranja* es un sistema destinado a automatizar el control del riego en entornos agrícolas. Su principal tarea es el control de la apertura y cierre de electroválvulas para permitir o cortar el paso de agua desde el sistema de riego. Dicho sistema permite la apertura o cierre inmediatos de las electroválvulas o la programación de las aperturas y cierres en distintas horas a lo largo de una semana.

Además, *smartgranja* cuenta con un sistema de control de usuarios (protegidos por contraseña) que permite limitar las operaciones que puede solicitar cada categoría de usuarios (como abrir o cerrar una válvula, cambiar los instantes programados de apertura o cierre, o cambiar los privilegios de un usuario).

Las categorías de usuarios disponibles son:

- administrador: puede realizar cualquier operación sobre válvulas, temporizadores o usuarios.
- acceso a válvulas y temporizadores: puede realizar cualquier operación sobre válvulas y temporizadores.
- acceso a temporizadores: solamente puede realizar operaciones sobre temporizadores y consultas sobre válvulas.
- solo lectura: solamente puede consultar el estado del sistema respecto a válvulas y temporizadores (pero no puede realizar modificación sobre ellos).

3.2. Análisis de requisitos

El análisis de requisitos es una parte fundamental de los procesos de ingeniería software. En él se desarrollan los requisitos del sistema. Para ello convertimos los requisitos del cliente en otro conjunto de ellos que definan «lo que el sistema debe hacer y cómo debe hacerlo».[DL01]

Los requisitos funcionales (FR) se pueden definir en dos puntos: describen qué debe hacer el sistema para completar una operación y deben ser «orientados a acción»[Kos+11]

Los requisitos no funcionales (NFR) describen otras características del sistema, por ejemplo: el tiempo de respuesta, disponibilidad (indica la parte del tiempo total en las que el usuario puede usar cualquier funcionalidad del sistema sin haber lugar a error), usabilidad de la interfaz de usuario, etc. Además, estos requisitos, en contraposición con los funcionales, pueden ser subjetivos y relativos: esto quiere decir que pueden ser interpretados de

forma diferente según quien lo analice y que pueden cambiar según el sistema al que estén dirigidos.[Ada15]

Los requisitos funcionales que podemos encontrar en nuestro sistema en base a la definición hecha previamente son los que aparecen en el apéndice A.1. Por su parte los no funcionales se reducen a los que aparecen en el apéndice A.2.

En el cuadro 3.1 podemos ver un ejemplo de como están expuestos los requisitos. En primer lugar aparece un indicador que es un nombre único para cada uno, en el del ejemplo aparece FR1 refiriéndose a «functional requirement 1». En la siguiente columna observamos la descripción del requisito. Y, finalmente, la importancia, este parámetro nos indica la importancia (subjetiva) que consideran (desarrollador y cliente) que tiene el cumplimiento del requisito.

Identificador	Descripción	Importancia
FR1	El sistema dará de alta las válvulas automáticamente, una vez que estas hayan sido arrancadas.	alta

Cuadro 3.1: Requisitos funcionales

3.3. Descripción del modelo de dominio

El modelo de dominio es una representación de la relación existente entre todas las entidades de la realidad existente en torno al problema para el que se va a diseñar una solución software y, a su vez, descriptiva de las propias entidades. El correspondiente modelo para el sistema *smartgranja* está expresado en el diagrama de clases UML de la figura 3.1.

Figura 3.1: Diagrama del modelo de dominio del sistema

Las clases, que serán descritas con posterioridad, son: Válvula y temporizador.

- Valvula: Componente físico que permite pasar o no el agua para el riego. Es dada de alta en el sistema por un usuario con privilegios suficientes.
- Temporizador: Clase que define el instante de tiempo de apertura de la válvula, el instante de cierre y el día o días de la semana a los que se aplica. Lo crea el usuario y está asociado a una válvula concreta.

Algunas de las relaciones posibles entre clases según permite la notación UML, son las siguientes:

- Asociación: Se expresa mediante una linea sólida e indica una relación existente entre dos clases, el nombre que aparece sobre la linea explica el significado de la asociación.

- Composición: se representa mediante una asociación con un rombo sólido en un extremo, e indica una relación del tipo todo-parte, donde la parte (representada por la clase que no está en contacto con el rombo) no puede existir independientemente del todo (clase contenedora próxima al rombo).
- Agregación: Se expresa mediante una asociación con un rombo hueco en un extremo, e indica una relación del tipo todo-parte, donde la parte (representada por la clase que no está en contacto con el rombo) sí puede existir independientemente del todo (clase contenedora pr'oxima al rombo).

3.4. Descripción de casos de uso

En los casos de uso, vistos desde un enfoque analítico (en lugar de uno de diseño), lo que podemos observar es la comunicación entre el usuario y el sistema, sin detallar nada sobre la comunicación entre las diferentes partes que conforman el sistema.

De lo comentado en párrafo anterior se puede deducir que el único actor que aparece en esta sección es el usuario que denominaremos «User» y que definimos como la representación de quien va a utilizar el sistema.

Estos casos de uso los podemos clasificar en tres grandes grupos. Aquellos que guardan relación con la gestión de usuarios: las figuras A.3, A.4, A.5, A.6 y A.7. Aquellos relativos a la gestión de válvulas: figuras A.8, A.9, A.10, A.11 y A.12. Y aquellos que están relacionados con la gestión de temporizadores, que corresponden a las figuras A.13, A.14, A.15, A.16 y A.17.

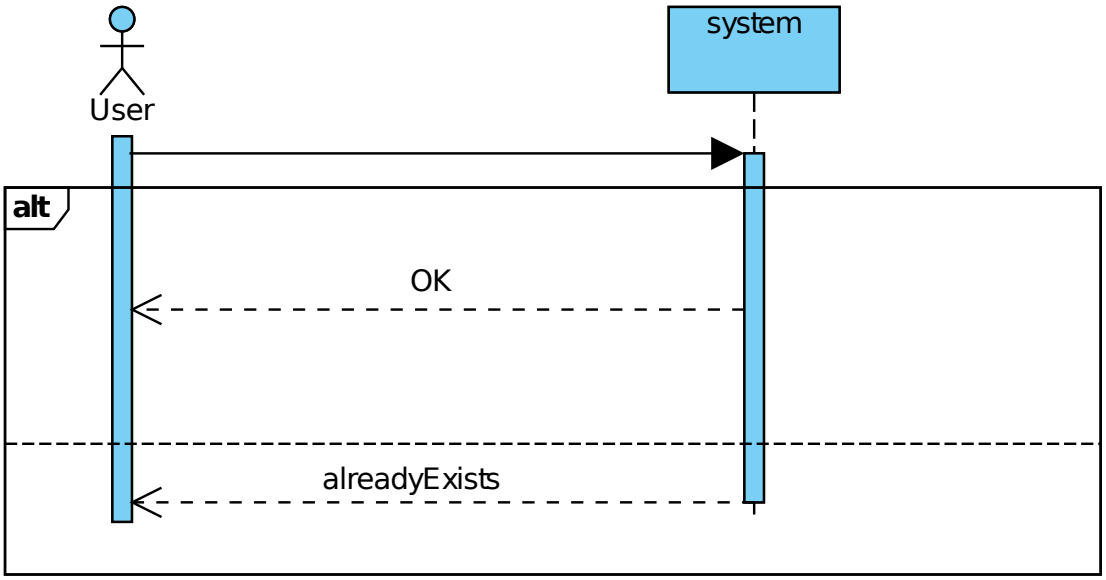

FIGURA 3.2: Análisis del caso de uso de creación de los usuarios

Capítulo 4

Diseño

4.1. Arquitectura hardware

Generalizando la definición de arquitectura software que después aplicaremos en la sección 4.2 (basada en [Bus01], la arquitectura hardware es la descripción de los componentes hardware de un sistema y la relación existente entre esos componentes.

4.1.1. Análisis de decisiones

El punto principal de nuestro sistema es, evidentemente, la activación y desactivación del riego. Por lo tanto es por ahí por donde se debe empezar este análisis. El riego en entornos agrícolas normalmente está íntimamente relacionado con conductos que transportan el agua desde una «llave de paso» hasta un difusor, aspersor, etc. Es justo ese conducto el punto en el que debe existir un componente que abra o cierre el paso de agua y que se pueda controlar electrónicamente.

Esa función la realizará la electroválvula (elemento que comentábamos en la sección 2.4). Por lo tanto, será el elemento principal de nuestro sistema.

A partir de aquí. Nuestro sistema debe permitir controlar las electroválvulas que se instalen desde la interfaz de usuario. Para ello debemos debemos contar con una aplicación de control que cumpla con los requisitos no funcionales, debe ser accesible desde ordenador, móvil, *tablets*,etc. y desde cualquier lugar (no necesariamente a una distancia próxima a la válvula).

La aplicación, por tanto (por cumplir esos requisitos no funcionales que le fuerzan a ser accesible a distancia de la propia electroválvula y controlada desde los dispositivos mencionados), por sí misma no va a ser capaz de controlar una electroválvula que necesita, recordemos, un pulso eléctrico cuadrado para cambiar de estado. Por lo tanto, se necesita un elemento que haga de controlador de la válvula y que se pueda comunicar con la aplicación web que proveerá la interfaz de usuario y la gestión del sistema.

El elemento o conjunto de elementos dedicados a esta función debe cumplir ciertas características, estas características son: ser capaz de enviar pulsos rectangulares de las características requeridas, presentar un consumo reducido y poder comunicarse mediante alguna tecnología inalámbrica de comunicación con la aplicación web. Esta última carácteristica se debe al cumplimiento de los requisitos de accesibilidad presentes en el cuadro de requisitos no funcionales (ver cuadro A.2).

Respecto a la tecnología de comunicaciones, debido a la distancia posible entre un ordenador que aloje la aplicación de control respecto al controlador de la válvula (dado que si fuera bluetooth para poder ser accesible desde cualquier parte sería necesario que estuviera disponible en un servidor cercano al controlador), se ve necesario que sea una aplicación se

comunique con el controlador mediante WiFi, lo que nos permitirá desplegar el servicio fácilmente independientemente de las características del lugar, simplemente añadiendo repetidores para aumentar la cobertura WiFi disponible.

Las placas de desarrollo Arduino, NodeMCU o las series LPC de NXP cumplen los requisitos que se acaban de enumerar. Además, es deseable que el elemento controlador de las válvulas pueda ser **alimentado mediante baterías**, dado que el controlador no debería necesitar estar conectado a la red eléctrica para evitar el cableado adicional como se comentaba anteriormente en la misma sección.

Finalmente, la tecnología escogida es la placa de desarrollo Arduino MKR WiFi 1010. Escogemos Arduino frente a NodeMCU debido a la cantidad de documentación disponible, a que es un sistema estable y debido a que las tres características fundamentales las cumple como hemos comentado previamente. La elección del modelo es debido a que elimina la necesidad de dispositivos adicionales para realizar la comunicación vía WiFi con la aplicación Web a desarrollar.

Características de la placa Arduino MKR WiFi 1010

La placa de desarrollo Arduino puede estar desconectada de la red eléctrica y recibir alimentación a través de dos entradas (para voltaje y para tierra). Se puede comprobar en su página web [Ard20b], este voltaje expresado como la diferencia entre la entrada previamente denominada «voltaje» y la denominada tierra (del ingles «ground») oscila, dependiendo del modelo, entre los 5 y 12v. En la referencia previa se indica que ese voltaje dependerá del modelo concreto de placa; por ejemplo para la placa MKR WiFi 1010 son 5v [Ard20a].

Uno de los factores que nos interesan en el proyecto es que la solución ayude a reducir la cantidad de trabajo (horas de trabajo de personal contratado, voluntario, propietario..) necesaria para controlar el riego. Y como, además, se explicaba anteriormente, se requiere que los controladores funcionen sin estar conectados a la red eléctrica, el consumo tiene que ser reducido para poder evitar que el cambio de las baterías supongan un trabajo diario o semanal adicional.

La placa Arduino consume alrededor de 150mA enviando y recibiendo paquetes WiFi y alrededor de 30mA en el modo de bajo consumo según su página web oficial [Ard20c].

La placa MKR WiFi 1010 contiene un módulo que le provee de la conectividad necesaria, este modulo se denomina «*u-blox NINA-W102*» [Ard20a]. Como se puede comprobar en la «hoja de datos» del módulo citado [uBl19], soporta implementa las normas WiFi n (IEEE 802.11n) exclusivamente en la banda de 2.4 GHz, y permite cifrado mediante WEP (del inglés *Wired Equivalent Privacy*) y WPA2 (del inglés *WiFi Protected Access 2*).

4.1.2. Esquema de la arquitectura hardware

La arquitectura hardware, por tanto, la podríamos describir así: un ordenador que aloje el servicio web que será consumido por un usuario a través de móvil, *tablet*, ordenador, etc. este servicio controlará, a su vez, varios «circuitos controladores» que serán los encargados de transformar las órdenes para las placas controladoras Arduino (algunas de esas órdenes se transformarán en las señales necesarias para abrir o cerrar algunas de las válvulas controladas).

En la figura 4.1, se muestran los elementos citados anteriormente unidos mediante lineas sólidas con o sin flechas. El sentido de la flecha indica cuál de las partes realiza las solicitudes (u operaciones sobre la otra parte) y cuál de las partes las recibe, por ejemplo: la flecha que une el «circuito controlador» y la válvula indica que el «circuito controlador» ejecutará operaciones sobre la válvula (y nunca ocurrirá en sentido opuesto).

Por sencillez visual, en la imagen se muestra una válvula correspondiente a cada «circuito controlador», pero hemos de tener en cuenta que cada uno de estos circuitos podrá gestionar las ordenes para una o varias válvulas (no necesariamente una).

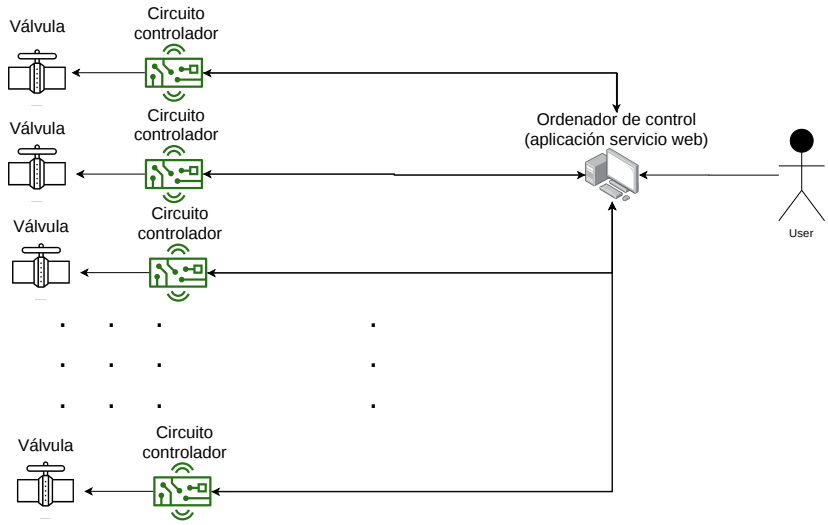

FIGURA 4.1: Esquema de arquitectura del hardware

4.1.3. Esquema de montaje controlador-electroválvula

La placa de desarrollo Arduino solo permite controlar voltajes de hasta 3.3V en sus «pines» digitales, esta amplitud no es suficiente para los pulsos cuadrados de una gran parte de las electroválvulas de corriente continua (que son las que utilizaremos para facilitar el desarrollo del sistema). Uno de los principales fabricantes a nivel internacional [Bir20] y otro fabricante con un volumen de ventas creciente en España [Rai20] proveen sus válvulas de corriente continua a 9V. Tampoco es complicado encontrar electroválvulas que funcionen a 12V en corriente continua.

Para amplificar la amplitud del pulso a 9-12V decidimos incluir placas de control auxiliares L298N en el montaje, el cual nos permite controlar la señal de salida en función de un voltaje de alimentación mediante dos señales de control y una de habilitación. Además, este módulo dispone de dos tríos de las entradas comentadas y dos pares de salidas, con lo cual podremos controlar dos electroválvulas por cada módulo utilizado.

El circuito controlador consiste en la placa de desarrollo Arduino, las baterías necesarias y la placas de control auxiliar L298N. En el esquema de la figura 4.2 podemos observar las conexiones que se deben hacer con la placa Arduino (aparece una placa «Arduino Uno» pero la numeración de los pines utilizados es la misma que en la placa «Arduino MKR WiFi 1010») para controlar dos válvulas. Todas las pilas que aparecen en la figura son de 1.5V.

Se pueden observar ciertas conexiones con un solo cable en el conector, el cual aparece en el extremos superior de la imagen. A cada una de esas bornes le corresponde una conexión

con su correspondencia en el circuito integrado L298N Estas correspondencias son de esta forma (todas las referencias al integrado L298N hacen referencia al esquema de la figura 4.3, donde aparece el conjunto de entradas y salidas de ese integrado):

- Borne con cable rojo en el extremo inferior: Le corresponde una conexión con la entrada del L298N señalada con un rotulo «+12V».
- Borne con cable rojo en la tercera posición desde abajo: Le corresponde una conexión con la salida del L298N rotulada con «+5V».
- Borne con un cable verde: Le corresponde una conexión con la entrada «IN1» del L298N.
- Borne con un cable amarillo: Le corresponde una conexión con la entrada «IN2» del L298N.
- Borne con un cable naranja: Le corresponde una conexión con la entrada «ENA» del L298N.
- Borne con un cable azul turquesa: Le corresponde una conexión con la entrada «IN3» del L298N.
- Borne con un cable marrón: Le corresponde una conexión con la entrada «IN4» del L298N.
- Borne con un cable rosado: Le corresponde una conexión con la entrada «ENB» del L298N.

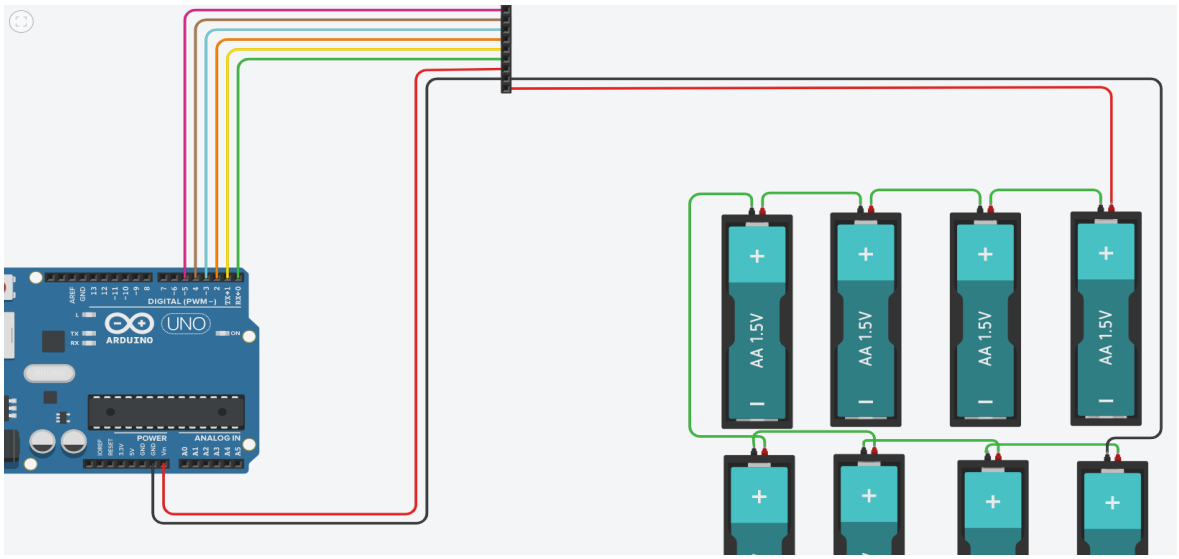

FIGURA 4.2: Esquema de montaje

Las conexiones con la válvula corresponderán a las salidas «OUT1» «OUT2» para una válvula, y «OUT3» y «OUT4» para la otra.

4.2. Arquitectura software

Para introducir el concepto de «arquitectura software» tomaremos la definición que se da en [Bus01], que traducida y resumida dice que una arquitectura software es la descripción de los componentes de un sistema software y la relación existente entre ellos.

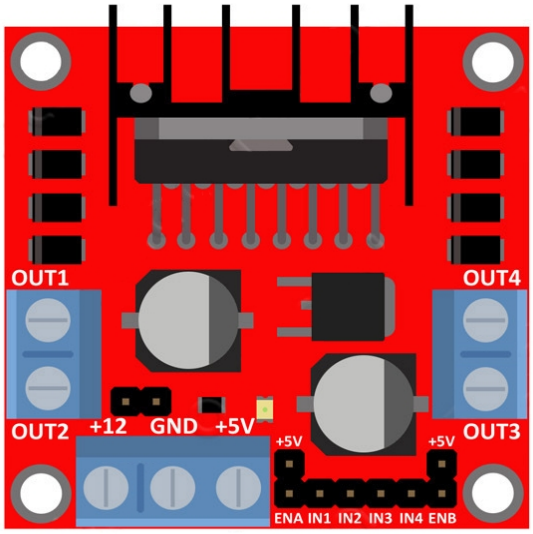

FIGURA 4.3: Circuito integrado L298N [Des20]

Según los requisitos no funcionales, la aplicación que proporcione la interfaz de usuario en el sistema, debe ser accesible desde ordenador, móvil, *tablets*.. y desde cualquier lugar (no necesariamente a una distancia determinada de la válvula).

La forma elegida de cumplir con estos requisitos es una aplicación web, dado que estas, si son desarrolladas utilizando un diseño adaptativo, son accesibles desde cualquier plataforma que disponga de un navegador web. Además, una aplicación móvil que esté disponible a través de un servidor web en una dirección IP pública puede ser accesible desde cualquier lugar con conexión a Internet (aunque internamente es preferible un enlace local para evitar tener la aplicación en un estado inaccesible debido a problemas con la conexión a Internet...).

El servicio web, cuya arquitectura se detallará más adelante, se encuentra alojada en la máquina que, en el esquema de arquitectura hardware 4.1, aparece con un rotulo que indica «ordenador de control». Este proporcionará una interfaz al usuario y se comunicará con la aplicación Arduino alojada en la placa de desarrollo Arduino en el «circuito controlador».

Este desarrollo software se divide en el desarrollo de dos subsistemas (los cuales se muestran en la figura 4.4), una aplicación que se ejecuta en cada placa Arduino y, una aplicación de control (denominada «servicio web *smartgranja*») que se ejecuta en el ordenador de control. Este último sigue la arquitectura ROA/D (véase la sección 2.1.2) y conforma un servicio web restful (véase la sección 2.1.3).

4.2.1. Servicio web *smartgranja*

En el servicio web se encuentran la vista del sistema, los recursos, los controladores de cada vista y la parte del modelo referente a datos. Este servicio web se ha desarrollado como un servicio web restful.

- En lo que respecta a la vista del sistema, en nuestro caso al usuario se le mostrarán las colecciones de válvulas y las válvulas solicitadas, las colecciones de temporizadores y los temporizadores solicitados, se recogerán sus ordenes, a saber, operaciones CRUD (crear, leer, actualizar y borrar) sobre válvulas, sobre temporizadores y sobre usuarios y se recogerá la información necesaria como algunos datos de la válvula o del temporizador.
- En el caso de *smartgranja*, cada vista se alcanza con una petición GET específica que será gestionada por en el recurso asociado a esa vista. El conjunto de estos conforman la

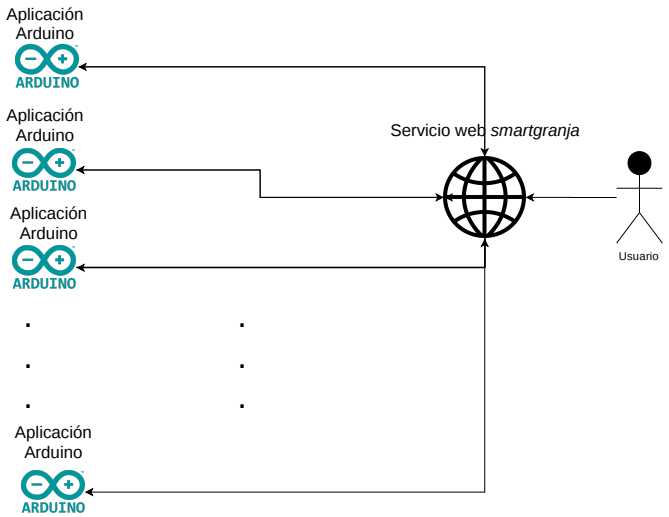

FIGURA 4.4: Esquema de la arquitectura software

capa que denominamos de recursos. Estos recursos proporcionan a la vista la información necesaria, para que sea mostrada al usuario y solicitan, a su vez, información del usuario a la vista. Los recursos pueden ejecutar peticiones HTTP sobre otros recursos. Estas peticiones HTTP se derivan al recurso en cuestión gracias a la URI (Uniform Resource Identifier) y el recurso conoce lo que se le solicita gracias a la operación (GET, POST, PUT...).

Estos recursos desglosan las acciones solicitadas por el usuario (o por otros actores del sistema) en acciones más simples encomendadas a los controladores.

En estos últimos se encuentra la lógica de negocio, aquí es donde se solicita al modelo el conjunto de operaciones necesarias para ejecutar las acciones que componen la respuesta del sistema a las solicitudes del usuario.

- Los controladores realizan las tareas fundamentales de la lógica de negocio de la aplicación y obtienen y guardan información persistente comunicándose con la capa de modelo.

- En la arquitectura software de nuestra aplicación la capa de modelo está implementada mediante una base de datos. La gestión de la base se realiza por parte de ciertos controladores, los cuales gestionarán sus accesos.

  Toda la información relativa a usuarios, categorías de usuarios y válvulas y temporizadores existentes en el sistema será almacenada en la base de datos. Esta capa debe responder a las solicitudes de datos por parte de los controladores. En secciones posteriores se mostrará esta relación con más detalle.

4.2.2. Aplicación Arduino

Aplicación Arduino como parte del sistema software completo

La aplicación Arduino forma, como parte del sistema software completo, un servicio restful con su propia interfaz que será consumida por los recursos del servicio web que se ejecuta en el ordenador de control, a través de la capa de controladores.

Presenta una arquitectura multicapa con una capa de recursos y una capa de controladores. No necesita una capa de vista porque no va a interactuar con ella directamente ningún usuario, ni necesita una capa de modelo porque no va a almacenar datos de una forma persistente.

4.3. Diseño del sistema

El sistema de *smartgranja* se centra en dos conceptos: los usuarios y las válvulas, el resto de los conceptos contenidos van a tener alguna relación con estos dos. Esto es debido a que son los dos elementos más importantes para la solución la requerida.

4.3.1. Descripción del modelo del diseño

El modelo del diseño del sistema *smartgranja* está expresado en el correspondiente diagrama B.1.

Contiene clases que representan los conceptos lógicos que conformarán las diferentes partes del sistema y la relación entre ellos.

Este modelo es similar al de modelo del dominio, dado que se pretende dar una respuesta software a la situación real (representada en el modelo de dominio). Por lo tanto, las clases del dominio comentadas previamente son trasladadas a clases software aquí presentadas. A esto se le añaden otras clases software que representan: a los roles, a los usuarios y a sus controladores.

Las clases que contiene el modelo son:

- Usuario: Persona que utilizará la aplicación de gestión del sistema de riego y que pertenecerá a alguno de los roles predeterminados del sistema.
- Rol: Clase que define los privilegios de cada usuario. Por lo tanto, cada usuario tiene asociado un rol.
- Administrador, ResourcesWrite, TimersWrite, ReadOnly: Roles a los que se les deben asociar los privilegios propios del rol correspondiente en el modelo del Dominio, a saber, los del cuadro B.1
- Válvula: Clase software que representa a la válvula física existente en el modelo del Dominio, contendrá información sobre el estado de la válvula física (abierta o cerrada).
- Temporizador: Clase software que define un momento concreto durante el cual una válvula está abierta. El comienzo y la finalización del mismo son atributos, como también el día o días en los que se aplicará.
- Controladores: Cada uno de ellos es una clase controladora (explicadas anteriormente en la sección 4.2.1) dedicada al control de una parte de la aplicación: válvulas (y temporizadores) y usuarios (y roles).

Los tipos de asociación que aparecen en el diagrama del modelo son los mismos explicados en la descripción del modelo del Dominio 3.3

4.3.2. Descripción de casos de uso

Para describir los casos de uso vamos a detallar con qué requisitos está relacionado, que actor inicia el caso y cómo es la comunicación que ocurre entre el actor y el sistema, ignorando lo que ocurre dentro del sistema. Estas descripciones aparecen en los cuadros 4.1 a 4.17.

Los actores que aparecen son:

- User: Actor que representa a la persona que vaya a utilizar el sistema.
- Arduino: Actor que representa a una de las partes del sistema (controlador terminal). Este actor actúa como tal únicamente contra el servicio web.

Los casos de uso a describir aparecen en el diagrama de casos de uso (ver figura B.2)

Nombre	Creación de usuarios
Requisitos relacionados	FR21
Actor que inicia el caso de uso	Usuario
Análisis de caso de uso	El actor «Usuario» solicitará al sistema la creación de un nuevo usuario. El sistema responderá creándolo de acuerdo a la información previamente proporcionada por el actor «Usuario» y almacenando la información.
Descripción completa	Figura B.3

Cuadro 4.1: Creación de usuarios

Nombre	Listado de usuarios
Requisitos relacionados	FR22
Actor que inicia el caso de uso	Usuario
Análisis de caso de uso	El actor «Usuario» solicitará el listado de todos los usuarios existentes en el sistema. El sistema responderá enviandole el listado completo de usuarios almacenados con información abreviada.
Descripción completa	Figura B.4

Cuadro 4.2: Listado de usuarios

Nombre	Visualización de usuarios
Requisitos relacionados	FR23
Actor que inicia el caso de uso	Usuario
Análisis de caso de uso	El actor «Usuario» puede solicitar la información de un determinado usuario. El sistema responderá mostrándole toda la información de que disponde relativa al usuario.
Descripción completa	Figura B.5

Cuadro 4.3: Visualización de usuarios

Los diseños de estos casos de uso están expresados con diagramas de secuencia en el apéndice B.2.3. Se hará referencia a ellos más adelante en la sección de implementación.

Nombre	Modificación de usuarios
Requisitos relacionados	FR24
Actor que inicia el caso de uso	Usuario
Análisis de caso de uso	El actor «Usuario» puede solicitar la modificación de un usuario introduciendo la nueva información. El sistema responderá efectuando la modificación en la información almacenada y mostrandole la información modificada al actor «Usuario».
Descripción completa	Figura B.6

Cuadro 4.4: Modificación de usuarios

Nombre	Borrado de usuarios
Requisitos relacionados	FR25
Actor que inicia el caso de uso	Usuario
Análisis de caso de uso	El actor «Usuario» puede solicitar el borrado de determinado usuario. El sistema debe borrar la información de ese usuario almacenada e inhabilitar su acceso al sistema.
Descripción completa	Figura B.7

Cuadro 4.5: Borrado de usuarios

Nombre	Creación de válvulas
Requisitos relacionados	FR1 - FR2 - FR18
Actor que inicia el caso de uso	Valve controller (controlador Arduino)
Análisis de caso de uso	En el momento instalación de las válvulas, el controlador Arduino iniciará el registro de la válvula en el sistema. El sistema ha de responder registrando la válvula correspondiente.
Descripción completa	Figura B.8

Cuadro 4.6: Creación de válvulas

Nombre	Listado de válvulas
Requisitos relacionados	FR8
Actor que inicia el caso de uso	Usuario
Análisis de caso de uso	El actor «Usuario» debe poder visualizar la lista completa de válvulas, cuando sea solicitado por el actor, el sistema responderá devolviendo la colección de válvulas con información abreviada de cada una de ellas.
Descripción completa	Figura B.9

Cuadro 4.7: Listado de válvulas

Nombre	Visualización de válvulas
Requisitos relacionados	FR6
Actor que inicia el caso de uso	Usuario
Análisis de caso de uso	El usuario debe poder visualizar la válvula, cuando sea solicitado por el usuario, el sistema responderá devolviendo la información completa de la válvula.
Descripción completa	Figura B.10

Cuadro 4.8: Visualización de válvulas

Nombre	Modificación de válvulas
Requisitos relacionados	FR7
Actor que inicia el caso de uso	Usuario
Análisis de caso de uso	El usuario debe poder solicitar la modificación de la información de la válvula almacenada, el sistema debe modificar esa información y mostrar al usuario la información solicitada.
Descripción completa	Figura B.11

Cuadro 4.9: Modificación de válvulas

Nombre	Borrado de válvulas
Requisitos relacionados	FR9
Actor que inicia el caso de uso	Usuario
Análisis de caso de uso	El actor «Usuario» solicitará la eliminación de una válvula. El sistema eliminará toda la información almacenada de la válvula y desactivará y eliminiará los temporizadores relacionados.
Descripción completa	Figura B.13

Cuadro 4.10: Borrado de válvulas

Nombre	Apertura y cierre de válvulas
Requisitos relacionados	FR4 - FR5
Actor que inicia el caso de uso	Usuario
Análisis de caso de uso	El actor «Usuario» solicita al sistema la apertura o cierre de la válvula. El sistema debe responder abriendo o cerrando la válvula e informando al usuario.
Descripción completa	Figura B.12

Cuadro 4.11: Apertura y cierre de válvulas

Nombre	Vencimiento del temporizador
Requisitos relacionados	FR3 - FR5
Actor que inicia el caso de uso	Tiempo
Análisis de caso de uso	El caso se inicia cuando vence el temporizador fijado, bien para el cierre o bien para la apertura. El sistema debe responder abriendo o cerrando la válvula y modificando la información almacenada sobre el estado de la misma.
Descripción completa	Figura B.14

Cuadro 4.12: Vencimiento del temporizador

Nombre	Creación del temporizador
Requisitos relacionados	FR11 - FR18
Actor que inicia el caso de uso	Usuario
Análisis de caso de uso	Este caso se iniciará cuando el actor «Usuario» solicite la creación de un temporizador asociado a una válvula con toda la información necesaria para un temporizador (hora de apertura, de cierre y día o días en que se repite la apertura y cierre), el sistema creará el temporizador, lo almacenará y devolverá al usuario la información del temporizador creado.
Descripción completa	Figura B.15

Cuadro 4.13: Creación del temporizador

Nombre	Modificación del temporizador
Requisitos relacionados	FR14
Actor que inicia el caso de uso	Usuario
Análisis de caso de uso	En este caso, el usuario solicitará la modificación de cierta información del temporizador. El sistema responderá haciendo la modificación necesaría en la información almacenada, modificando las posteriores aperturas o cierres de la válvula y devolviendo la información modificada al usuario.
Descripción completa	Figura B.18

Cuadro 4.14: Modificación del temporizador

Nombre	Listado de los temporizadores
Requisitos relacionados	FR13
Actor que inicia el caso de uso	Usuario
Análisis de caso de uso	El actor «Usuario» solicita la lista completa de temporizadores asociados a una válvula concreta. Una vez solicitada por parte del usuario, el sistema devolverá la colección completa de temporizadores con información abreviada de cada temporizador.
Descripción completa	Figura B.16

CUADRO 4.15: Listado de los temporizadores

Nombre	Visualización del temporizador
Requisitos relacionados	FR12 - FR18
Actor que inicia el caso de uso	Usuario
Análisis de caso de uso	El usuario debe poder solicitar la información de un temporizador concreto, el sistema responderá devolviendo la información completa de ese temporizador.
Descripción completa	Figura B.17

CUADRO 4.16: Visualización del temporizador

Nombre	Borrado del temporizador
Requisitos relacionados	FR15
Actor que inicia el caso de uso	Usuario
Análisis de caso de uso	El usuario solicita el borrado de un temporizador concreto. El sistema desactivará las aperturas y cierres de la válvula configurados de acuerdo a ese temporizador y eliminará el temporizador almacenado.
Descripción completa	Figura B.19

CUADRO 4.17: Borrado del temporizador

4.3.3. Diseño del servicio web *smartgranja*

El servicio web es un servicio restful planteado para poder ser accedido mediante un navegador web y para ser robusto, simple y escalable.

- La parte de **modelo** del servicio web está conformada por clases que podemos deducir analizando el diagrama de modelo de diseño B.1.

  Viendo el diagrama podemos observar las clases: «Rol», «Usuario», «Válvula», «Temporizador». Las clases de dominio «Administrador», «ResourcesWrite», «TimersWrite» y «ReadOnly» hacen referencia a tipos de roles, por tanto instancias de esa clase con diferentes atributos.

  Los atributos que aparecen en el diagrama son los que descubrimos en el momento de realizar el análisis del caso al que estamos proponiendo una solución. Los atributos pueden ampliarse desde el punto de vista de diseño en el que ya hemos tomado ciertas decisiones que les pueden afectar. A continuación se detallan los atributos necesarios desde el punto de vista del diseño.

  - Usuario
    - Nombre: Nombre de la persona a la que se le asigna el usuario.
    - Login: Identificador único de usuario utilizado en el inicio de sesión
    - Password: Contraseña utilizada para porteger la cuenta y utilizada en el inicio de sesión
  - Rol
    - Nombre de rol: Nombre único que identifica a un rol
    - Privilegios sobre temporizadores
    - Privilegios sobre válvulas
  - Válvula
    - Id de válvula: Nombre único que identifica a la válvula compuesto por la MAC del controlador y un identificador local de la válvula dentro del controlador.
    - Nombre de la válvula: Cadena que introduce el usuario para ayudarle a identificar más fácilmente el recurso.
    - Nombre de la parcela: Nombre de la parcela de terreno donde se encuentra la válvula.
    - Coordenadas: Coordenadas GPS para identificar en un mapa la válvula.
  - Temporizador
    - Id del temporizador: Nombre único (en el dominio de temporizadores de una válvula) que identifica al temporizador .
    - Hora de comienzo: Hora en la que la válvula debe abrirse.
    - Hora de finalización: Hora en la que la válvula debe cerrarse.
    - Días de repetición: Días en los que debe repetirse esta secuencia.

- **La vista** del servicio web coincide con la vista del sistema *smartgranja*.

  Está implementado mediante la utilización de los lenguajes java y HTML, el framework freemarker de java y la librería bootstrap para HTML.

  Esta capa está contenida enteramente en las plantillas freemarker, dado que en estas plantillas se especifica la forma en la que los datos van a ser presentados y en que las

acciones van a ser ofertadas al usuario, así mismo las plantillas solicitan al recurso que las haya invocado los datos necesarios para mostrar al usuario.

Aquí vemos una de las separaciones funcionales de la arquitectura multicapa, a saber la vista muestra los datos que se han de presentar al usuario solicitándoselos a los controladores.

En el sistema *smartgranja*, necesitamos una vista que dé la posibilidad al usuario de solicitar el inicio de la ejecución de los casos de uso de los que él mismo es el actor, y observar los resultados que el sistema le deba ofrecer.

Tomando los casos de uso (que podemos encontrar en el apéndice B.2.3) y escogiendo solo aquellos en los que el usuario aparece como actor nos quedamos con: Creación de usuarios, listado de usuarios, visualización de usuarios, modificación de usuarios, borrado de usuarios, listado de las válvulas, visualización de la válvulas, modificación de la válvula, apertura y cierre directo, borrado de la válvula, creación de temporizadores, listado de temporizadores, visualización de temporizadores, modificación de temporizadores y borrado de temporizadores.

De todos ellos necesitamos generar una vista solo de los que entran en estos tres tipos (las vistas como resultado de cualquier otro será la de alguno de estos). Uno de ellos sería la respuesta a los listados (de usuarios, de válvulas y de temporizadores), otro de ellos es la respuesta a las peticiones de visualización de un elemento completo (usuario, válvula o temporizador), los diseños de estos casos de uso pueden verse en la sección B.2.3. Aparte de estas plantillas, es necesaria una para la gestión gráfica de ciertos errores y una última para la visión inicial de la aplicación.

- **Los recursos** (clases que componen la capa de recursos comentada previamente), atendiendo a la tarea de «interpretación de las solicitudes de ejecución de acciones que realiza» que le encomendábamos en la sección 4.2, divide las acciones solicitadas por los actores en otras más simples que solicita a los controladores.

  Cada uno de estos recursos está asociado a una URL del servicio REST que la aplicación java ofrece para ser consumida por los navegadores y por el controlador de las válvulas. A su vez, cada recurso define el manejo de la respuesta a las operaciones REST que se ejecuten sobre el URL. Las operaciones GET con content-type HTML están planteadas para ofrecer las vistas (plantillas HTML con los datos requeridos del recurso) a los navegadores que lo soliciten. Al tener, el sistema, un caso de uso de visualización por cada uno de los recursos y colecciones, va a existir un recurso por cada una de las vistas.

  Esta relación que comentamos entre recurso, URL, y operación REST, es la que podemos observar en el diagrama de recursos del servicio web (ver figura B.20). En este diagrama podemos encontrar esta relación para todos los recursos que se van a implementar en el servicio web: el recurso raíz (Root), la colección de válvulas (Valves), los elementos de esa colección (Valve), la colección de temporizadores (Timers) y sus elementos (Timer), el recurso de gestión (Admin), la colección de usuarios (Users) y sus elementos (User) y un último recurso para el registro de los mensajes de información del controlador hacía la aplicación (LoggedEvents).

  Basándonos en esto y observando los casos de uso definidos en la sección B.2.3, que muestran las peticiones necesarias para la comunicación en cada caso, podemos concluir la siguiente relación entre controladores, clases java de los recursos y peticiones manejadas por los controladores (Véase cuadro 4.18).

- Los controladores, podríamos decir que ofrecen interfaces a los recursos, lo que permite, a estos últimos, abstraerse de la forma en la que modificar ciertos aspectos de más

Controller	Resource/Collection	Requests
UsersResourceController	Users collection	GET - HTML POST - HTML
UserResourceController	User	GET - HTML PUT - HTML DELETE - HTML
ValvesResourceController	Valves collection	GET - HTML
ValveResourceController	Valve	GET - HTML PUT - JSON PUT - HTML PATCH - JSON PATCH - HTML DELETE - HTML
TimerResourceController	Timers collection	GET - HTML POST - HTML
TimersResourceController	Timer	GET - HTML PUT - HTML DELETE - HTML

CUADRO 4.18: Relación entre controlador, recurso y peticiones admitidas

bajo nivel (la forma de crear registros en una base de datos, abrir o cerrar conexiones con ellas...). Esto disminuye las tareas de escalabilidad y mantenimiento, dado que una pequeña modificación en la forma de de realizar una de estas acciones, no implicará más que ese pequeño cambio en el controlador correspondiente.

En la figura 4.5 podemos ver el modelo de diseño del servicio web, donde se observa la estructira de esta capa de controladores. En este diseño aparecen dos controladores para gestionar las dos principales entidades del diseño en el servicio; los usuarios y las válvulas. De la gestión de esas dos entidades se deriva el resto de la gestión, la de los temporizadores por el hecho de estar contenidos en las válvulas y la de los roles por ser necesarios y estar incluidos en los usuarios.

Las relaciones observadas entre temporizadores y válvulas (composición) y entre usuarios y roles (agregación) se deben a que; por un lado, los temporizadores son una entidad que pierde sentido cuando la válvula desaparece del sistema y que deben ser eliminados cuando la válvula sea eliminada, y por otro lado, los usuarios necesitan tener un rol asociado y el rol preexiste al usuario.

Por tanto, la comunicación entre la vista y el controlador se basa en que la vista solicita datos al navegador por medio de las variables incluidas en la plantilla, el controlador incluye esas variables y estas, conjuntamente con la plantilla son devueltas como respuesta a la petición del navegador.

4.3.4. Diseño de la base de datos

Entidades

Las entidades que conformarán la base de datos que necesitamos en nuestro sistema son 4.

Estas 4 entidades (y sus atributos) son algunas de las las clases que aparecen descritas en el modelo en la sección de diseño del sistema web. Que sean estas se debe a que son los datos

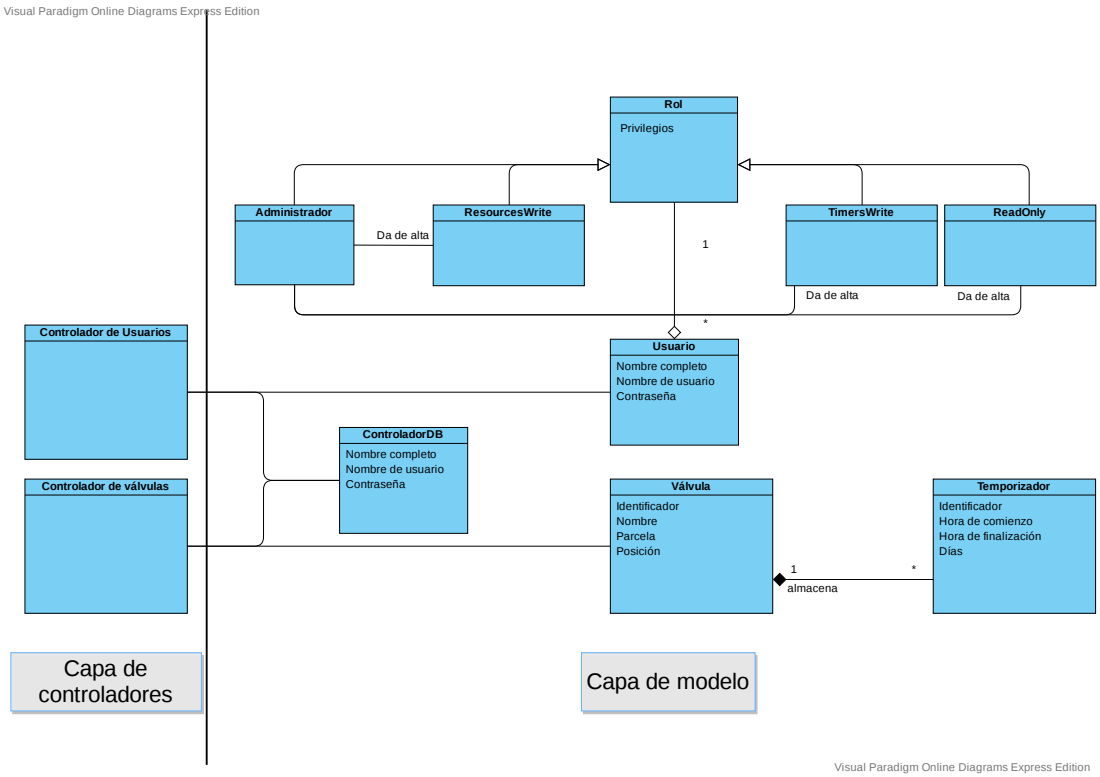

FIGURA 4.5: Diagrama de modelo de diseño de la aplicación java

que van a ser gestionados por la aplicación java y que tendrán que permanecer invariables en cada inicio de la aplicación Enumerándolas, son: Válvulas, Temporizadores, Usuarios y Roles.

Relaciones

El siguiente paso en el diseño de la base de datos es deducir las relaciones que deben existir entre las diferentes entidades.

En nuestro caso las relaciones que deben existir son:

- Válvulas - Temporizadores: Relación uno a varios donde la cardinalidad mínima para la entidad válvulas es de 1 y de 0 para la entidad temporizadores, y las cardinalidades máximas son 1 para las válvulas y * (no hay máximo y puede haber más de uno) para los temporizadores.
- Usuarios - Roles: Relación uno a varios donde la cardinalidad mínima para la entidad usuarios es de 0 y de 1 para la entidad roles, y las cardinalidades máximas son 1 para la entidad roles y * (no hay máximo y puede haber más de uno) para los usuarios.

Entre las entidades válvulas y usuarios o válvulas y roles no existe relación, lo mismo ocurre entre las entidades temporizadores y usuarios o temporizadores y roles.

Claves

Cada una de las 4 entidades conforma una tabla en la base de datos, cada una de estas tablas contiene una clave primaria. Estas claves son: Id de válvula para la tabla de válvulas, una clave compuesta por Id de válvula e Id de temporizador para la tabla de temporizadores,

el campo «nombre de usuario» en la tabla de usuarios y el campo «nombre del rol» en la tabla de roles.

Las claves cumplen las tres reglas formales; valor único de cada registro, campos no clave dependiendo completamente de la clave y y no hay ningún campo con dependencia de otro que no sea clave.

Finalmente la relación entre las entidades y claves queda como aparece expresada en el diagrama relación-entidad (figura 4.6).

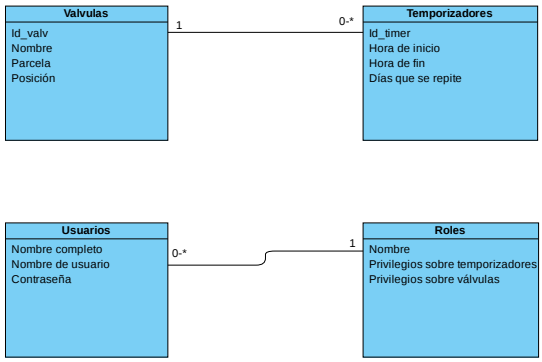

FIGURA 4.6: Diagrama relación-entidad

4.3.5. Diseño de la aplicación Arduino

La aplicación que se ejecuta en las placas Arduino también se ha desarrollado como una aplicación de tipo servicio web restful

El modelo contará con un conjunto de válvulas y otros conjuntos de temporizadores contenidos en cada válvula. Estas clases del modelo no podrán actuar directamente sobre sus respectivos componentes físicos (lo cual implicaría que todo lo relativo a la válvula incluyendo su apertura y cierre este contenido en la válvula y que ocurra lo mismo con el temporizador) debido sobretodo a dos limitaciones;

- Arduino no permite, debido a una limitación de la placa, hacer pulsos durante las rutinas de interrupción. Lo cual nos obliga a extraer la apertura y cierre fuera de la rutina y por lo tanto separarlo del cambio de estado del objeto lógico.
- La placa de desarrollo no permite fijar más de una alarma (elemento que utilizaremos para fijar el comportamiento de la placa de acuerdo a la información contenida en los temporizadores) simultáneamente. Esto nos impide poder gestionar la activación y

desactivación desde la propia clase «Temporizador», dado que necesitaremos apoyarnos en otras clases que crearemos «ad hoc», para permitir la gestión de los múltiples temporizadores en múltiples válvulas disponiendo de una sola alarma.

Como desarrollábamos en la sección de arquitectura de la aplicación Arduino 4.2.2, el usuario de la aplicación Arduino es el servicio de control diseñado previamente, y que se ejecuta en el ordenador principal. De esta manera, cada uno de los conjuntos operación-URL (combinación de una operación HTTP con un URL; por ejemplo: operación GET a /root/valves/") conllevará la invocación de un método de un controlador del servicio de control principal (dicho método también se encargará de crear y enviar la respuesta correspondiente a la petición HTTP).

Esto nos conduce a un proceso de diseño en el que primero diseñamos el servicio web restful y después cada una de las funciones del controlador asociadas a REST.

REST

Analizando los casos de uso del sistema expuestos en la sección de diseño del sistema 4.3.2, podemos deducir las operaciones y URL que la aplicación Arduino deberá poner a disposición del servicio web de control, para el correcto funcionamiento del sistema. La forma de hacer esta deducción es observar, en los diagramas de secuencia de los casos de uso, la comunicación entre el servicio web de control o cualquiera de las partes del sistema y el servicio web que se ejecuta en las placas Arduino.

De esta forma la relación entre casos de uso del sistema, URL del servicio web restful de Arduino y operaciones sobre esos URL es la que aparece en la tabla 4.19.

Por motivos de seguridad, el servicio web restful debe desautorizar todas las solicitudes de operación no contempladas e impedir las conexiones de máquinas no autorizadas.

Funciones asociadas

Todas las funciones asociadas deben contener tres partes, una en la que se analizan los parámetros necesarios de la petición, otra en la que se realiza la acción correspondiente y una última en la que se gestiona la respuesta.

Las acciones (la acción central que comentamos en el párrafo previo, entre el análisis de la petición y la gestión de la respuesta) que se deben llevar a cabo en cada una de las funciones, las deducimos analizando la parte relacionada con Arduino de los diagramas de secuencia de los casos de uso asociados a la función. En la tabla 4.19 podemos ver una breve descripción de cada una de estas acciones.

Funciones del controlador no asociadas al servicio web restful

Dar de alta las válvulas: debe disponer de una función que se invoque cada vez que la placa Arduino se inicie, que implemente la parte del caso de uso B.8 referente a la placa Arduino. Para ello debe solicitar la creación de la válvula a la aplicación de control. Si la aplicación de control responde que la válvula ya existe, entonces la aplicación Arduino tendrá que solicitar el listado de temporizadores y, en caso de haberlos, solicitar uno a uno todos los existentes (para fijar sus características y determinar el instante del próximo evento de activación y desactivación).

Gestión de temporizadores

A modo de simplificación, en los apartados previos hemos hablado acerca de una activación de alarma para referirnos a la configuración de una interrupción del sistema en un momento concreto.

En la presente sección, citábamos una limitación de la placa de desarrollo; era referida a la imposibilidad de configurar dos de estas interrupciones simultáneamente.

Ante esta imposibilidad, diseñamos este método de gestión de los temporizadores y las interrupciones temporales. Este se basa en tener siempre configurada la interrupción disponible con el próximo comienzo o final de temporizador.

Cada temporizador dispone una hora de comienzo y otra de fin como se indica en la sección de modelo del diseño. Cada vez que se cree un temporizador, se deberá comprobar si la hora de comienzo es la hora (de las que contienen los temporizadores) más próxima al momento actual, en el caso de que así sea, la interrupción temporal se debe configurar con esa hora. La modificación de temporizadores tiene las mismas implicaciones respecto a la gestión que la creación.

Dispondremos de dos rutinas de interrupción diferenciadas; una de apertura y otra de cierre. Cuando las rutinas de interrupción se activen, el controlador debe comprobar a qué temporizador corresponde y, así, comprobar para qué días está configurado el temporizador. En caso de coincidencia, debe solicitar la apertura o cierre correspondiente en la válvula (con la actuación correspondiente a nivel físico y la modificación del estado a nivel lógico), comprobar cuál es el comienzo o fin con distancia temporal más corta respecto al momento actual, y activarlo la correspondiente interrupción temporal.

Caso de uso	**URL**	**Operación**	**Acción por parte del controlador**
Apertura y cierre de válvulas	«/valves/ {localValveId}/»	PUT	Modificar el estado de las válvulas lógicas y enviar el pulso de apertura o cierre a la válvula
Borrado de la válvula	«/valves/ {localValveId}/»	DELETE	Desactivar todos los temporizadores asociados a la válvula, eliminar todos esos temporizadores y eliminar la válvula lógica.
Crear el temporizador	«/valves/ {localValveId}/ timers/»	POST	Crear el temporizador lógico, activar las alarmas correspondientes y devolver el identificador local del temporizador creado. Debido a la limitación comentada previamente (no se pueden activar varias alarmas simultáneamente), la gestión de los temporizadores es más compleja de lo citado aquí (no se deben activar inmediatamente). Por lo tanto, posteriormente se explicará como se realiza esta gestión 4.3.5.
Visualización del temporizador	«/valves/ {localValveId}/ timers/ {localTimerId}»	GET	Recuperar el temporizador lógico que se ha solicitado (buscando entre los de la válvula solicitada) y devolverlo al solicitante de la operación
Modificación de un temporizador	«/valves/ {localValveId}/ timers/ {localTimerId}»	PUT	Modificar la información del temporizador para que coincida con la proporcionada en la petición y actualizar la alarma en caso de que sea necesario.
Borrado de un temporizador	«/valves/ {localValveId}/ timers/ {localTimerId}»	DELETE	Desactivar la alarma correspondiente y borrar el temporizador lógico. Como en el otro caso citado, la alarma gestión de la alarma se especifica más ampliamente en la sección 4.3.5

CUADRO 4.19: Relacion entre casos de uso, servicio web restful de Arduino y controlador de Arduino

Capítulo 5

Implementación y pruebas

5.1. Implementación

La forma de realizar la implementación ha sido utilizando un desarrollo iterativo e incremental, en el que cada fase del ciclo de vida tiene su correspondencia con alguna parte de las pruebas indicadas en el plan de pruebas 5.2.1.

Comenzamos la implementación con la parte software de la aplicación que va a utilizar el usuario, por lo tanto, comenzamos con el servicio web y la base de datos. Concretamente, empezamos por el nivel de abstracción más bajo, a saber, la base de datos y el modelo.

La base de datos se ha implementado utilizando Hibernate y DerbyDB (DerbyDB como gestor de la base de datos e Hibernate para establecer una correspondencia entre cada tabla de la base de datos y una clase java, de forma que podamos gestionar cada registro de la base de datos como un objeto en java).

Posteriormente, se desarrollaron los controladores cuyo diseño se ha comentado previamente y que, gráficamente se puede observar en la figura 4.5. Entendimos en el proceso de implementación que la gestión directa de la base de datos por dos controladores diferentes incurre en una pequeña perdida de escalabilidad y mantenibilidad y, por ello, introdujimos otro controlador visto por los dos anteriores y que ofrece, a estos, una interfaz para el registro, recuperación, eliminación y actualización de elementos en la base de datos.

El siguiente paso fue desarrollar los dos servidores REST (el de la aplicación java y el de la aplicación Arduino); el de Arduino se desarrolló utilizando la librería AWOT y el de java utilizando el framework Restlet.

Finalmente se desarrollaron las diferentes vistas de la aplicación (utilizando la biblioteca Bootstrap y las plantillas de freemarker) y las diferentes clases (en java) y funciones (en Arduino) de manejo de las respuestas de los servidores REST. Para el desarrollo de la vistas, además de desarrollar plantillas de freemarker, hay que utilizar la clase de java de Freemarker «TemplateRepresentation» asociada para rellenar los datos de esas plantillas y devolver la plantilla como respuesta a la petición HTML.

5.2. Pruebas

5.2.1. Plan de pruebas

«Un plan de pruebas describe pruebas para los proyectos de desarrollo (...). En la planificación influyen (...) el alcance de las pruebas, los objetivos, los riesgos, los requisitos, la criticidad, la comprobabilidad y la disponibilidad de recursos».[cPU18]

El plan de pruebas, como parte del proceso de desarrollo software, es necesario para evitar que los posibles fallos existentes aparezcan cuando el software esté distribuido y sea detectado por el cliente.

Durante el proceso de desarrollo del sistema *smartgranja* se han ejecutado tres niveles de pruebas, atendiendo a su objetivo y momento de ejecución con respecto al proceso de desarrollo.

- **Objetivos:** Asegurar el funcionamiento correcto de la aplicación, el cumplimiento de los requisitos y la prevención de fallos inesperados

- **Riesgos:** Los riesgos más importantes que se han de cubrir son los de seguridad (acciones no autorizadas sobre el sistema). Otros riesgos importantes que hemos de tener en cuenta son los posibles efectos sobre los cultivos de un mal funcionamiento del sistema.

- **Requisitos:** Los requisitos a asegurar son los definidos en el proceso de análisis (requisitos funcionales A.1 y requisitos no funcionales A.2).

- **Alcance de las pruebas:** Las funcionalidades probadas van a ser las de los diferentes casos de uso, ya definidos en la sección de diseño. Pese a que estas sean las funcionalidades que se van a probar, no es lo único que se va a probar. También se probarán los diferentes subsistemas de forma aislada (aplicación java y aplicación Arduino), además de algunos de los componentes que los conforman: controladores de la aplicación java y servidor REST de la aplicación Arduino.

- **Estrategia de las pruebas:** Las pruebas se van a diseñar basadas en funcionalidades. Esto implica dividir el sistema en un conjunto de funcionalidades y, para cada una de ellas, diseñar pruebas que se encarguen de asegurar un funcionamiento adecuado tanto en escenarios positivos como negativos. Escenarios positivos son aquellos en los que el resultado esperado de la prueba es es la realización completa de la acción, mientras que los negativos son aquellos de los que se espera que la acción no sea realizada (por motivos de seguridad, de prevención de riesgos...). A cada una de las pruebas se le asignará un nivel de prioridad de acuerdo con la criticidad y el riesgo que comporte en caso de funcionamiento inadecuado.

  Además, se utilizará un estrategia de regresión que consistirá en realizar un conjunto grande de pruebas extremo a extremo periódicamente, y antes de cada despliegue. Estas pruebas tienen por objetivo asegurar que todas las funcionalidades, clases, etc., cuyo funcionamiento estaba comprobado sigue siendo el adecuado después de realizar modificaciones en el software para implementar nuevos requisitos o corregir errores detectados previamente.

- **Organización de las pruebas:** Las pruebas se van a ejecutar en diferentes fases en función del progreso del ciclo de trabajo del proceso de desarrollo. Las pruebas unitarias se van a ejecutar durante el desarrollo de bloques específicos de código, concretamente durante la implementación de las diferentes clases (solo se utilizarán para los controladores, no para todas las clases). Las pruebas de integración se van a ejecutar después de la integración de la implementación de un nuevo caso de uso con el resto del código (o de la integración de la modificación de un caso de uso existente). Las pruebas de aceptación se van a ejecutar previamente al despliegue de cada nueva versión de la aplicación e inmediatamente después.

Pruebas unitarias

Este tipo de pruebas consiste en probar de forma aislada un componente, este puede ser una clase, un método... Vamos a ejecutar este tipo de pruebas en algunas etapas del desarrollo para asegurar el funcionamiento de los controladores, el servidor REST y la aplicación Arduino, mientras se están desarrollando.

Lo hacemos de esta manera dado que la implementación de los controladores es previo al desarrollo completo, incluyendo todas las partes involucradas en el sistema, de cualquiera de los casos de uso y de la interfaz y, por tanto, es la única forma de asegurar que estas clases se han desarrollado de forma correcta. Ocurre lo mismo en el caso del servidor REST, aunque se podría probar con la interfaz de usuario que si estará en desarrollo, pero esa forma de probarlo introduciría muchas posibles fuentes de error.

Para la ejecución de estos casos de prueba de forma manual, se va a crear una interfaz muy simple consistente en solicitudes de datos en «la línea de comandos». El diseño de estas pruebas no aparece en la sección 5.2.1, dado que en su mayoría serán pruebas muy simples basadas en escenarios positivos y todas ellas diseñadas en el momento previo a la ejecución.

A modo de ejemplo del diseño de una prueba unitaria tenemos la tabla 5.1, donde cada sección de la tabla tiene el mismo significado que las expresadas en la sección 5.2.1. La implementación de este tipo de pruebas, para su ejecución manual, consistiría en dotar a estos controladores de una interfaz simple, consistente en una interacción a través de la línea de comandos, donde con un número se pueda seleccionar el método deseado y el controlador muestre el resultado de la misma forma.

Aspecto	**Especificación**
Nombre	Prueba acerca de lectura de una válvula en el controlador de válvulas
Tipo de prueba	Unitaria
Criticidad	Media
Precondiciones	Existe una base de datos con una tabla válvulas Existe una válvula en la base de datos con ID «id», nombre «name» y nombre de parcela «parcelName»
Escenario	El usuario introduce el número de válvula cuando le sea solicitado.
Postcondiciones	Los atributos de la válvula devueltos coinciden con los de la válvula deseada.

Cuadro 5.1: Ejemplo de prueba unitaria del controlador de válvulas

Pruebas de integración

La base de nuestras pruebas de integración va a ser el subsistema de la aplicación java y los diferentes casos de uso. Lo hacemos de esta manera para cumplir con el plan establecido donde se establece la estrategia de pruebas a seguir.

Cada conjunto de pruebas de integración desarrollado tienen en común la funcionalidad probada, por ejemplo la creación de una válvula, el listado de los temporizadores existentes, etc. Estas pruebas se ejecutarán de forma manual. Además el conjunto de las pruebas de una funcionalidad, deben asegurar el cumplimiento de todos los requisitos que guarden relación

con la funcionalidad y prevenir cualquier posible error, para ello se han diseñado escenarios negativos que preveen posibles acciones que deban concluir con un error ó un rechazo por parte del sistema.

Los diseños de estas pruebas aparecen en la sección 5.2.1, para buscarlos hemos de buscar en la fila «Tipo de prueba» y escoger aquellas en las que aparezca «integración».

Además de estas pruebas, también hemos de considerar como pruebas de intregación, las del servidor web restful de la aplicación Arduino (esta sería la base de las pruebas), que realizamos con un cliente REST externo (para aislar el objeto probado y que no dependiese de otros elementos en desarrollo) u utilizando la salida de Arduino utilizando el puerto serie disponible. El diseño de estas pruebas (las relativas al servidor web restful de la aplicación Arduino) no aparece con el conjunto de pruebas de la sección 5.2.1 Como ejemplo del diseño de una de estas pruebas del servidor web restful de la aplicación Arduino, tenemos la tabla 5.2, donde cada sección de la tabla tiene el mismo significado que las expresadas en la sección 5.2.1.

Aspecto	**Especificación**
Nombre	Prueba sobre la visualización de un temporizador en el servidor REST de Arduino
Tipo de prueba	Integración
Criticidad	Media
Precondiciones	Existe un temporizador en la placa de desarrollo Arduino en ese instante.
Escenario	Se realiza una petición GET al endpoint /valves/{valveId}/timers/{timerId} del servidor http://ip:puerto. Donde la variable «valveId» representa el identificador de la válvula donde se encuentra el temporizador, «timerId» representa el identificador del temporizador, «ip» representa la dirección IP de la placa de desarrollo Arduino y «puerto» el puerto configurado en la placa en el que atenderá (por defecto 80).
Postcondiciones	Los atributos del temporizador devueltos coinciden con los del temporizador deseado.

CUADRO 5.2: Ejemplo de prueba de integración del controlador de válvulas

Pruebas de aceptación

Con las pruebas de aceptación probamos, no solo las funcionalidades aisladas como en el caso de de las pruebas de integración, sino las funcionalidades en conjunto. Esto implica la necesidad de probar el funcionamiento de todos los subsistemas simultáneamente, por lo tanto, la base de nuestras pruebas de aceptación será el sistema completo.

Los diseños de estas pruebas aparecen en la sección 5.2.1, para buscarlos hemos de buscar en la fila «Tipo de prueba» y escoger aquellas en las que aparezca «aceptación».

Conjunto de pruebas

En la tabla 5.3 se agrupan las diferentes pruebas (según la funcionalidad probada en sus diseños) en diferentes conjuntos de pruebas. El nombre de cada conjunto hace referencia al caso de uso probado.

Conjunto	Pruebas
Listado de usuarios	Las pruebas de los cuadros C.1 y C.2
Creación de usuarios	Las pruebas de los cuadros C.3, C.4 y C.5
Visualización de los usuarios	Las pruebas de los cuadros C.6, C.7 y C.8
Modificación de usuarios	Las pruebas de los cuadros C.9, C.10 y C.11
Borrado de usuarios	Las pruebas de los cuadros C.12 y C.13
Listado de válvulas	La prueba del cuadro C.14
Visaulización de válvulas	Las pruebas de los cuadros C.15 y C.16
Modificación de válvulas	Las pruebas de los cuadros C.17, C.18 y C.19
Borrado de válvulas	Las pruebas de los cuadros C.20, C.21 y C.22
Creación de temporizadores	Las pruebas de los cuadros C.24, C.25, C.26 y C.27
Visualización de temporizadores	Las pruebas de los cuadros C.28 y C.29
Modificación de temporizadores	Las pruebas de los cuadros C.30, C.31 y C.32
Borrado de temporizadores	Las pruebas de los cuadros C.33, C.34 y C.35
Activación de los controladores	Las pruebas de los cuadros C.36 y C.37

CUADRO 5.3: Conjuntos de pruebas

Los cuadros de las pruebas muestran diferentes aspectos de la prueba:

- Nombre: El nombre es una frase descriptiva del escenario.
- Tipo de prueba: Indica los tipos de prueba a los que se puede aplicar la prueba (integración o aceptación).
- Criticidad: Según sea alta, baja o media, indica cuan crítico es el correcto funcionamiento del objeto de la prueba. Así en la criticidad de una prueba cuyo nombre sea «Creación exitosa de la válvula» nos indicará la importancia que tiene para el sistema que se pueda realizar correctamente la creación de la válvula.
- Precondiciones: Las condiciones que se deben de dar, previamente a la ejecución del escenario, para que se pueda realizar la prueba.
- Escenario: Una descripción de las acciones que se deben dar durante la ejecución.
- Postcondiciones: Las condiciones que se deben dar, al finalizar la ejecución, para que se considere exitosa la prueba.
- Ejemplos («atributo»): Cada una de las líneas que aparezcan asociadas a este aspecto implica una ejecución diferente. Cada una de estas ejecuciones debe fijar el valor del atributo «atributo» al indicado en la columna «Especificación»

5.2.2. Pruebas de campo

Posteriormente a los despliegues, se han hecho pruebas similares a la suma de los conjuntos de pruebas de aceptación, eliminando las de los conjuntos de usuarios y administración, en el escenario real de implantación del sistema.

Para hacer estas pruebas, hemos conectado la electroválvula junto con nuestro sistema a un canal de riego, en un punto intermedio entre una llave de paso (elemento que permite el control del paso de agua) manual y la salida del agua hacia el terreno. De esta forma, dejando abierta esa llave de paso, todo el control del paso de agua recae sobre la electroválvula e indirectamente sobre el sistema que la controla.

El mayor de los problemas que hemos encontrado, ha sido utilizar una electroválvula que, por razones desconocidas para nosotros, no cumplia la función que se esperaba de ella (abrir y cerrar la válvula como respuesta a los impulsos).

Descubrir que el error se encontraba en la electroválvula y no en el sistema fue cuestión de revisar y asegurar el funcionamiento de cada parte de los subsistemas del sistema global y, finalmente, comprobar si la salida del sistema cumplía con las especificaciones requeridas por la electroválvula (ancho de pulso y amplitud). Con otro dispositivo Arduino, y utilizando la entrada analógica de la que dispone, medimos el las características previamente mencionadas en el pulso que llegaba a la electrovaálvula. De esta forma, descartada la posibilidad de error del sistema, descubrimos que el error se encontraba en la electroválvula. Y sustituyéndola por otra repetimos las pruebas.

El resultado de estas pruebas ha sido satisfactorio, logrando un sistema funcional que cumple con todos los requisitos en el entorno real de la implantación definitiva.

Capítulo 6

Conclusiones y líneas futuras de trabajo

6.1. Conclusiones

Finalmente, se ha logrado diseñar e implementar la solución buscada con un sistema basado en una aplicación web que se encuentra alojada en un servidor apache con acceso desde la red interna y desde Internet por lo que puede ser accedida desde cualquier ubicación en la que se disponga de acceso a Internet. Esta aplicación se comunica mediante un servidor web «restful» con una placa de desarrollo Arduino, que a su vez controla un circuito L298N y, este genera los pulsos enviados a las electroválvulas conectadas a él.

Además de las pruebas unitarias, de integración y de aceptación que comprueban el funcionamiento de los diferentes componentes software, de las funcionalidades software y del sistema completo y que, están definidas en el plan de pruebas, se han ejecutado las pruebas de aceptación en un entorno agrícola real, con diversos tipos de válvulas.

Se ha comprobado el tiempo de respuesta desde que se ordena la activación de la válvula hasta que se activa la válvula realmente en condiciones de escasa cobertura WiFi. Estos tiempos eran de apenas 2 segundos, mientras que en condiciones de mayor intensidad de cobertura WiFi eran muy inferiores. Así mismo, el tiempo de respuesta desde el instante en el que el vence la alarma de arranque de una válvula en un temporizador es de aproximadamente 8 segundos, lo cual para el riego de terrenos agrícolas no es significativo.

6.2. Líneas futuras de trabajo

- Cambiar la tecnología de comunicación utilizada por una de bajo consumo que aumente el tiempo que transcurre entre cada cambio de las baterías y, que aumente el alcance de la red WiFi actual (sin la necesidad de utilizar repetidores WiFi que también tendrán que ser alimentados mediante baterías).

- Crear un servidor de automatización convencional, como puede ser «Jenkins», una configuración y una tarea adecuados para ejecutar los despliegues de cada nueva versión de forma adecuada. Por ejemplo, si se modifica el modelo de la base de datos, se debería crear un *script* que traslade los datos de la vieja estructura a la nueva. En este ejemplo, con una tarea de «Jenkins» se podría detener el servicio web, mover las bases de datos al directorio raíz del programa java que realizará la tarea, después podría mover las bases de datos resultantes al directorio original y, finalmente, iniciar el servicio web de nuevo.

- Añadir un sensor de humedad conectado a alguna de las entradas digitales de la placa Arduino con un divisor de tensión, de manera que la aplicación web reciba los valores leídos y los interprete para que el usuario pueda conocer, como parte de la monitorización, el estado de humedad del terreno.
- Podría añadirse un sensor de temperatura en el terreno (de forma similar a la expuesta para el sensor de humedad), de manera que el usuario conozca la temperatura aproximada del terreno en cada momento que lo requiera.
- Con ambos sensores (humedad y temperatura), se podría añadir una funcionalidad para que a elección del usuario el sistema elija de forma automática los temporizadores, recibiendo, por parte del ususario, la humedad relativa que el terreno debe mantener aproximadamente.
- Modificar la aplicación de control y la aplicación Arduino para facilitar la creación de temporizadores cuya hora de comienzo sea en un día diferente a la hora de finalización.
- Minimizar el trabajo necesario para mantener con carga las baterías, con las que se alimenta el circuito L298N e, indirectamente, la placa Arduino (pues su entrada de alimentación está conectada a una salida del circuito L298N). Por ejemplo, añadiendo placas solares que ayuden a recargar, durante las horas de luz, la batería gastada durante las horas sin luz, ó realizando modificaciones que reduzcan el consumo energético del circuito controlador. Actualmente, en el prototipo instalado, las baterías duran aproximadamente 3 días.

Apéndice A

Documentos de Análisis

A.1. Requisitos

A.1.1. Requisitos funcionales

Identificador	Descripción	Importancia
FR1	El sistema dará de alta las válvulas automáticamente, una vez que estas hayan sido arrancadas.	imprescindible
FR2	El sistema debe asociar a cada válvula con su conjunto de temporizadores permanentemente.	imprescindible
FR3	El sistema abrirá y cerrará las válvulas físicamente cada vez que llegue el momento marcado por el temporizador.	imprescindible
FR4	El sistema abrirá o cerrará las válvulas inmediatamente, cuando lo solicite un usuario con privilegios suficientes (escritura de válvulas).	imprescindible
FR5	El sistema modificará la información acerca del estado en que se encuentra la válvula en cada ocasión que se cierre o se abra, para proveer al usuario de información completamente actualizada.	imprescindible
FR6	El sistema mostrará la información de las válvulas a los usuarios con privilegios suficientes (lectura de válvulas) que lo soliciten.	imprescindible
FR7	El sistema permitirá la modificación de las válvulas a ususarios con privilegios suficientes (escritura de válvulas).	imprescindible
FR8	El sistema listará las válvulas disponibles a los usuarios con privilegios suficientes (lectura de válvulas) que los soliciten.	imprescindible
FR9	El sistema borrará las válvulas cuando sea solicitado por un usuario con privilegios suficientes (escritura de vávlulas).	imprescindible
FR10	El sistema borrará los temporizadores asociados a una válvula, de la que haya sido solicitado el borrado por un usuario con privilegios suficientes, y de la válvula misma.	imprescindible
FR11	El sistema creará los temporizadores que sean solicitados por usuarios con privilegios suficientes.	imprescindible
FR12	El sistema mostrará la información de un temporizador existente a los usuarios que lo soliciten.	imprescindible

Identificador	Descripción	Importancia
FR13	El sistema listará los temporizadores existentes a los usuarios que lo soliciten con privilegios suficientes (lectura de temporizadores)	imprescindible
FR14	El sistema modificará los temporizadores, previamente creados, a petición de usuarios con privilegios suficientes.	imprescindible
FR15	El sistema permitirá los usuarios, con permisos suficientes (escritura de temporizadores), borrar los temporizadores existentes.	imprescindible
FR16	El sistema deberá disponer de los diferentes roles enumerados en el cuadro B.1	alta
FR17	El sistema deberá realizar una gestión de permisos o privilegios para que cada usuario vea limitadas sus posibilidades de operación sobre recursos a lo correspondiente al rol asociado a su usuario, siguiendo lo indicado en los requisítos descritos anteriormente.	alta
FR18	El sistema permitirá a un usuario con rol asociado «administrador» ver las opciones de gestión de usuarios.	alta
FR19	*smartgranja* proveerá un sistema de gestión de usuarios.	alta
FR20	Cada usuario tendrá asociado un único rol.	alta
FR21	El nombre de usuario debe ser único	imprescindible
FR22	El sistema impedirá que un usuario sin el rol «administrador» asociado acceda a las opciones de gestión de usuarios.	alta
FR23	El sistema permitirá que los usuarios con rol «administrador» asociado creen otros usuarios.	alta
FR24	El sistema permitirá que los usuarios con rol «administrador» asociado visualicen todos los usuarios existentes.	alta
FR25	El sistema permitirá que los usuarios con rol «administrador» asociado visualicen todos los usuarios existentes.	alta
FR26	El sistema permitirá que los usuarios con rol «administrador» asociado visualicen todos los usuarios existentes.	alta
FR27	El sistema permitirá que los usuarios con rol «administrador» asociado visualicen todos los usuarios existentes.	alta

Cuadro A.1: Requisitos funcionales

A.1.2. Requisitos no funcionales

Identificador	Descripción	Importancia
NFR1	El sistema informático debe poder ser controlado desde cualquier dispositivo habitual (ordenador, tablet o móvil).	alta
NFR2	El sistema debe ser accesible desde cualquier lugar, no exclusivamente desde la red local.	alta

Cuadro A.2: Requisitos no funcionales

A.2. Modelo de dominio

FIGURA A.1: Diagrama de casos de uso del sistema

A.3. Casos de uso de análisis del sistema

A.3.1. Definición de actores de análisis

- User: Actor que representa a la persona que va a acceder, a través de un navegador web, a la aplicación que se ejecuta en el ordenador de control.

A.3.2. Diagrama de casos de uso de análisis

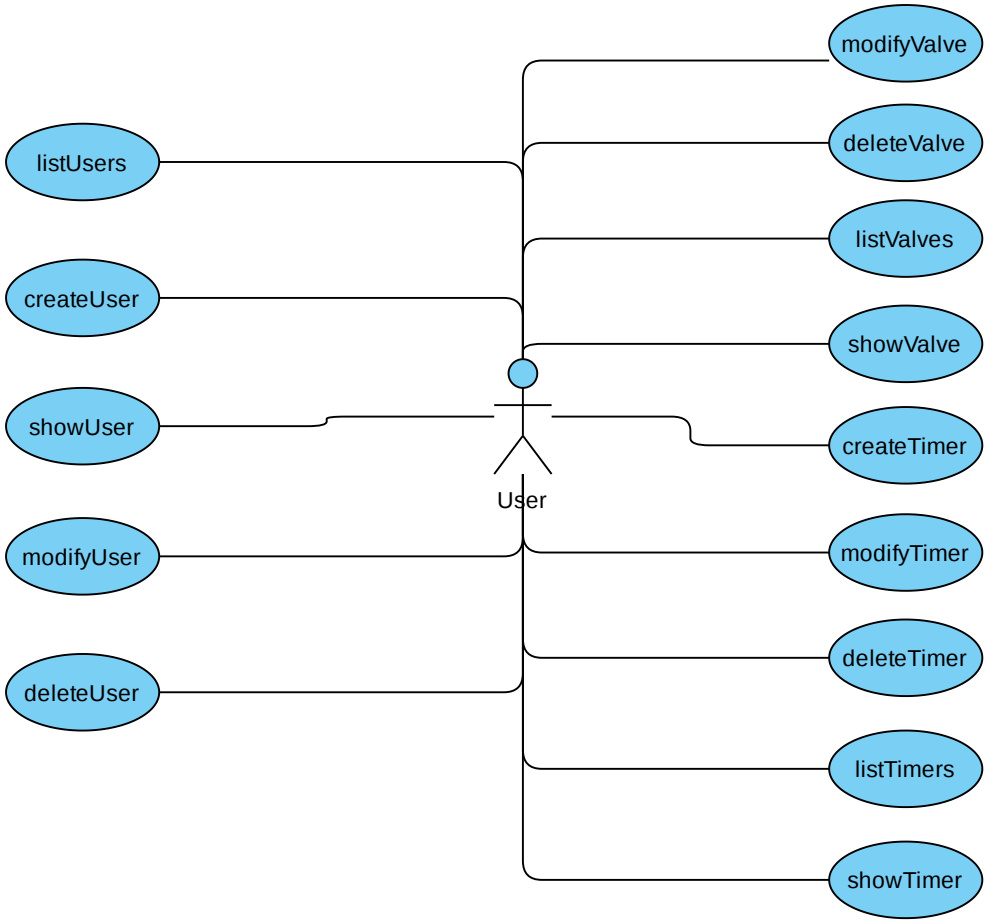

FIGURA A.2: Diagrama de casos de uso de análisis

A.3.3. Diagramas de secuencia de análisis del sistema

Creación de usuario

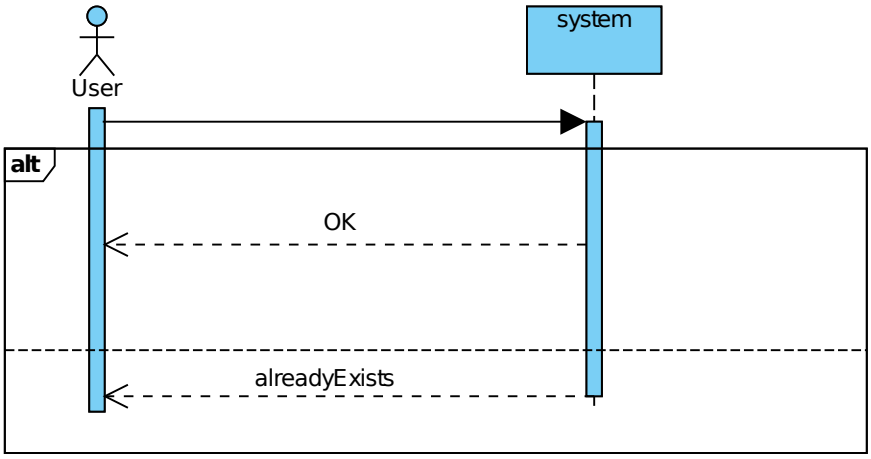

FIGURA A.3: Análisis del caso de uso de creación de un usuario

Listado de usuarios de la aplicación

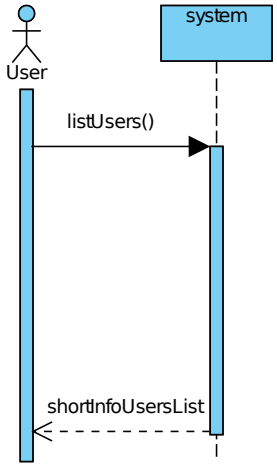

FIGURA A.4: Análisis del caso de uso de listado de los usuarios

Visualización de usuario

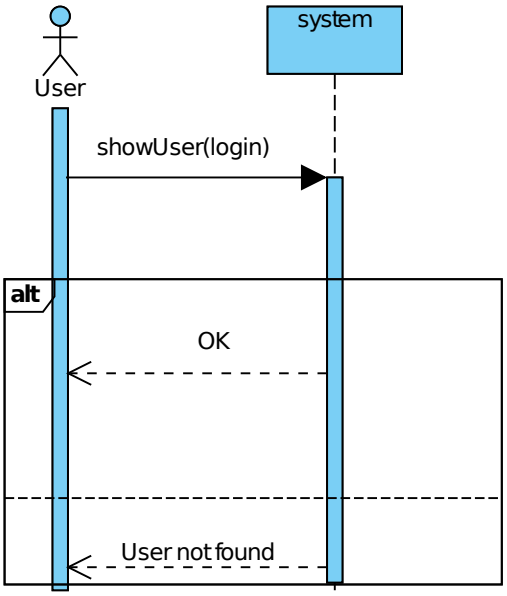

FIGURA A.5: Análisis del caso de uso de visualización de un usuario

Modificación de usuario

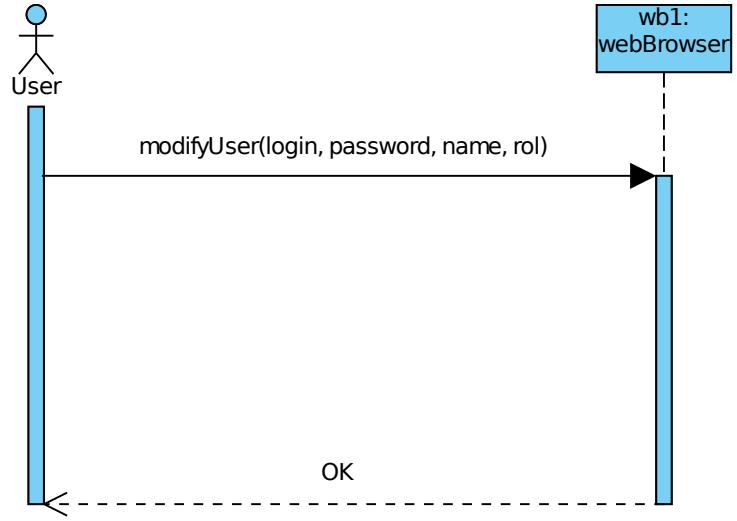

FIGURA A.6: Análisis del caso de uso de modificación de un usuario

Borrado de usuario

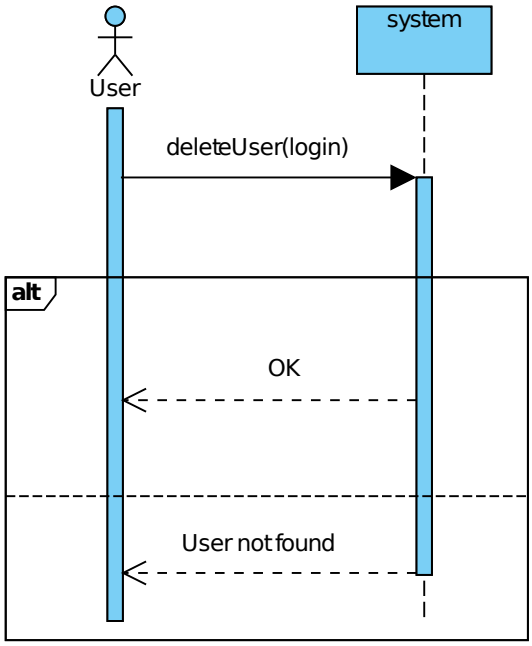

FIGURA A.7: Análisis del caso de uso de borrado de un usuario

Listado de válvulas

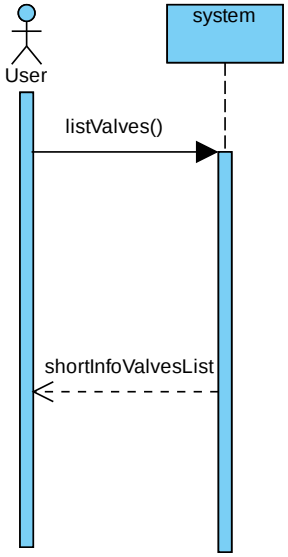

FIGURA A.8: Análisis del caso de uso de listado de las válvulas

Visualización de la válvula

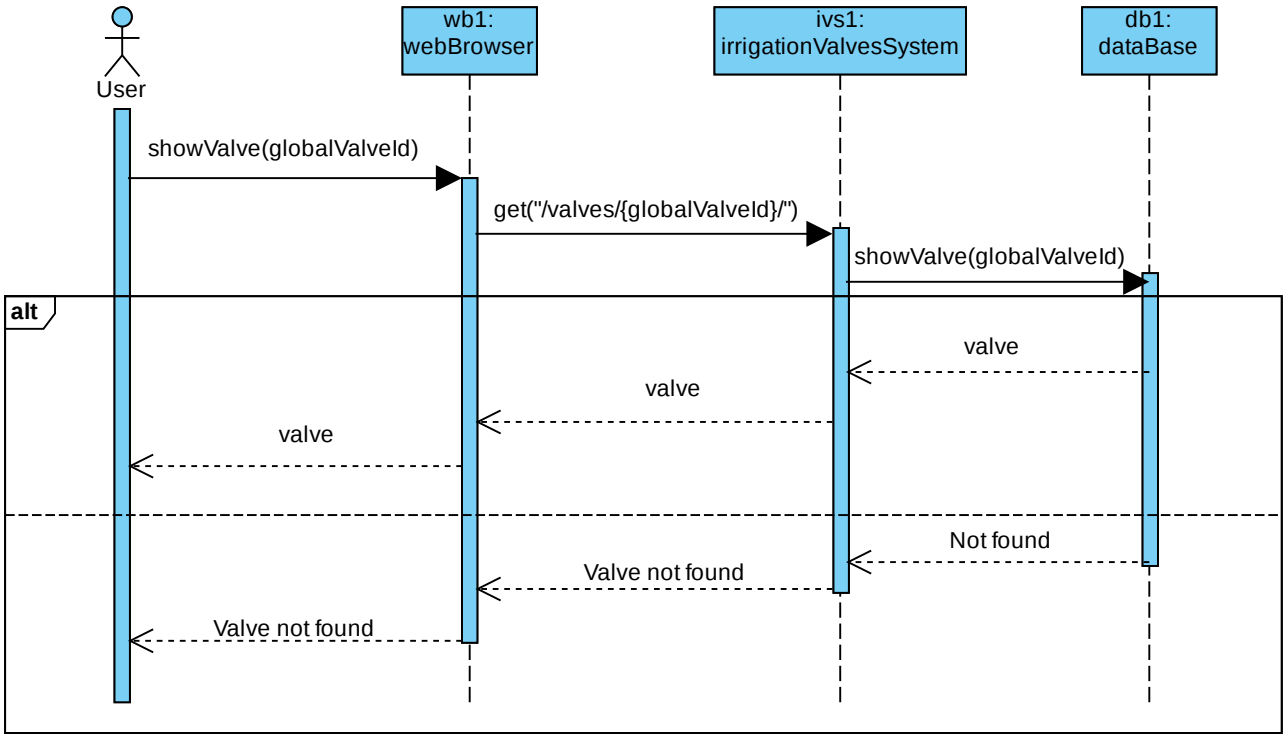

FIGURA A.9: Análisis del caso de uso de visualización de la válvula

Modificación de válvula

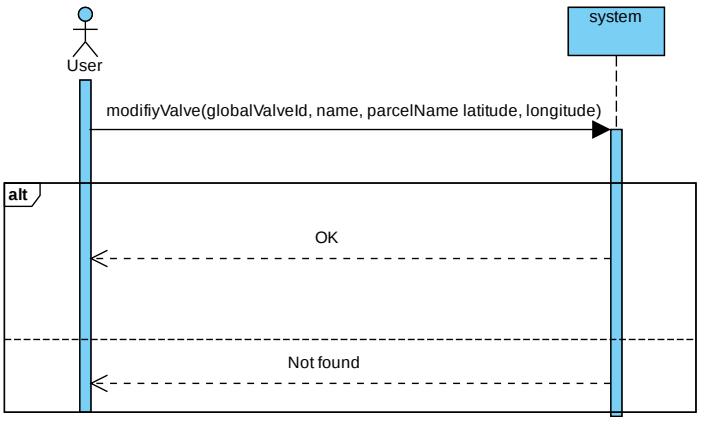

FIGURA A.10: Análisis del caso de uso de modificación de válvula

Apertura y cierre de válvula

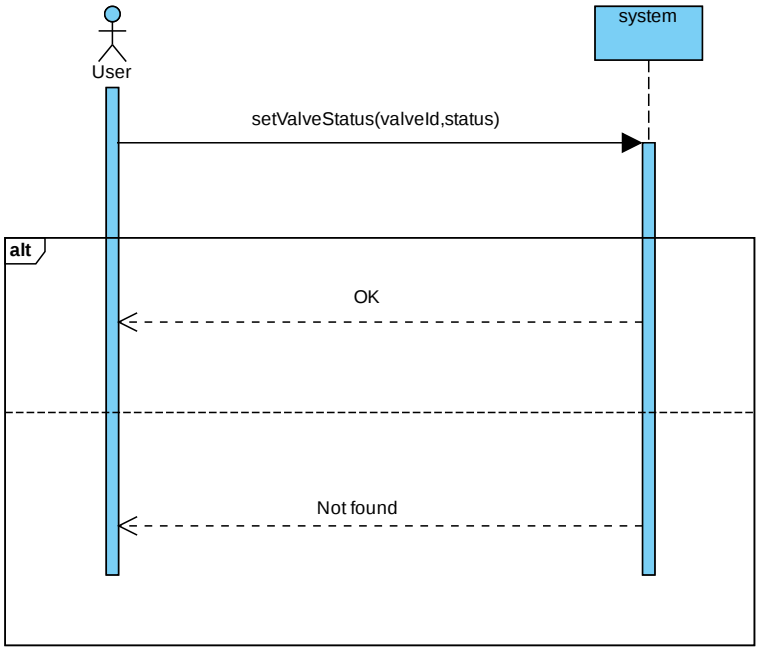

FIGURA A.11: Análisis del caso de uso de apertura y cierre de válvula

Borrado de válvula

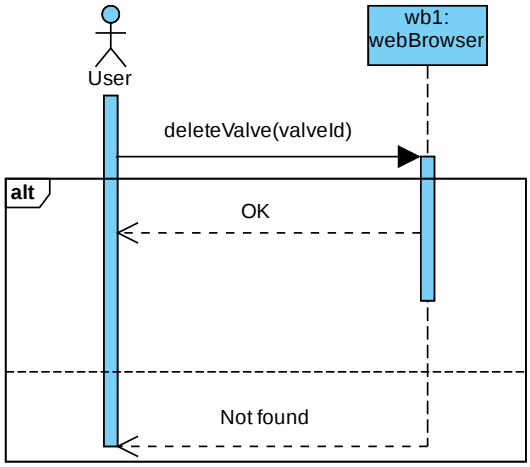

FIGURA A.12: Análisis del caso de uso de borrado de válvula

Creación de temporizador

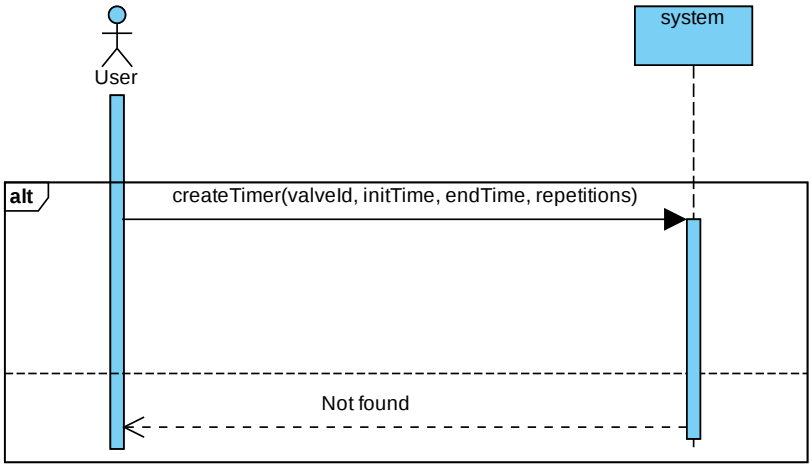

FIGURA A.13: Análisis del caso de uso de creación de temporizador

Listado de temporizadores

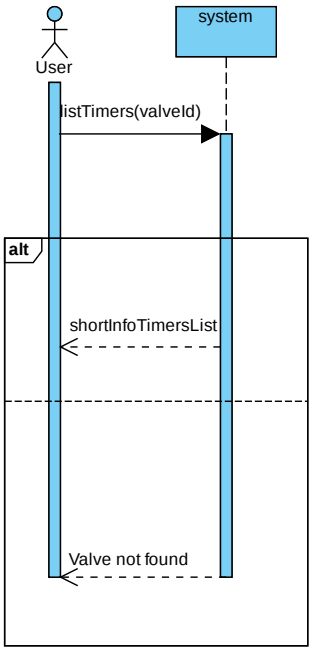

FIGURA A.14: Análisis del caso de uso de listado de temporizadores

Visualización de temporizador

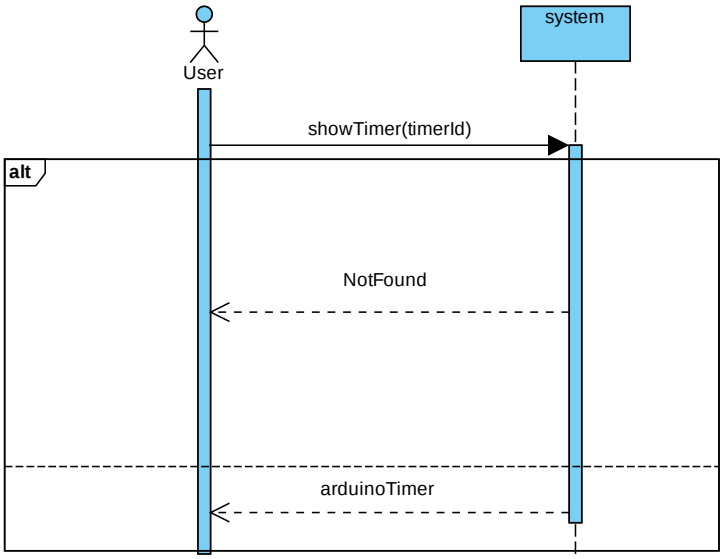

FIGURA A.15: Análisis del caso de uso de visualización del temporizador

Modificación de temporizador

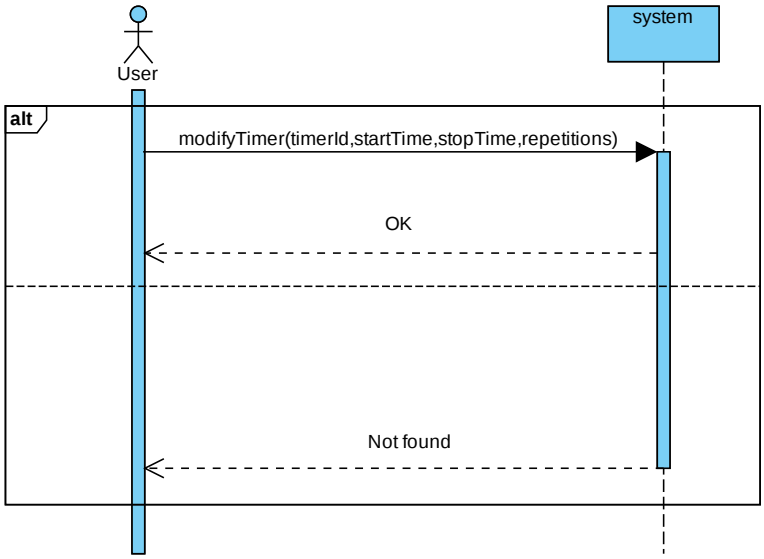

FIGURA A.16: Análisis del caso de uso de modificación de temporizador

Borrado de temporizador

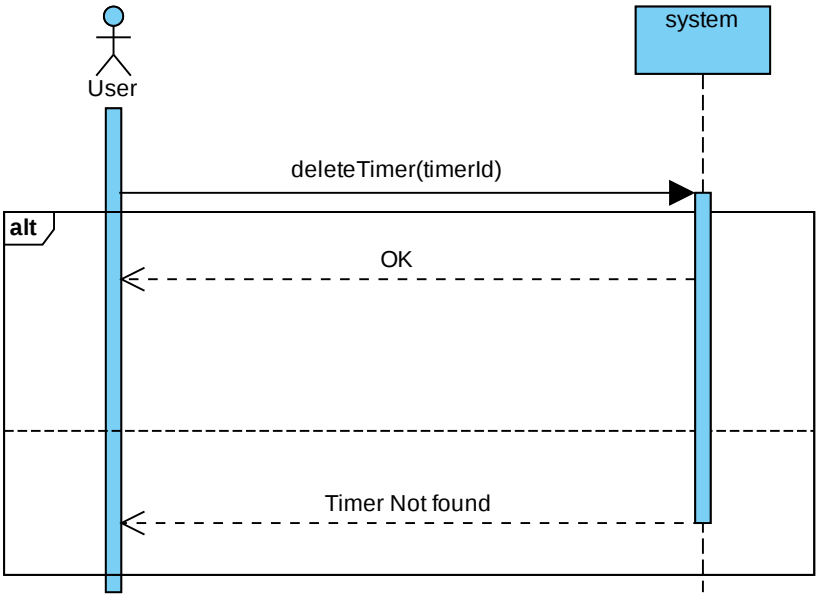

FIGURA A.17: Análisis del caso de uso de borrado de temporizador

Apéndice B

Documentos de diseño

B.1. Modelo del diseño

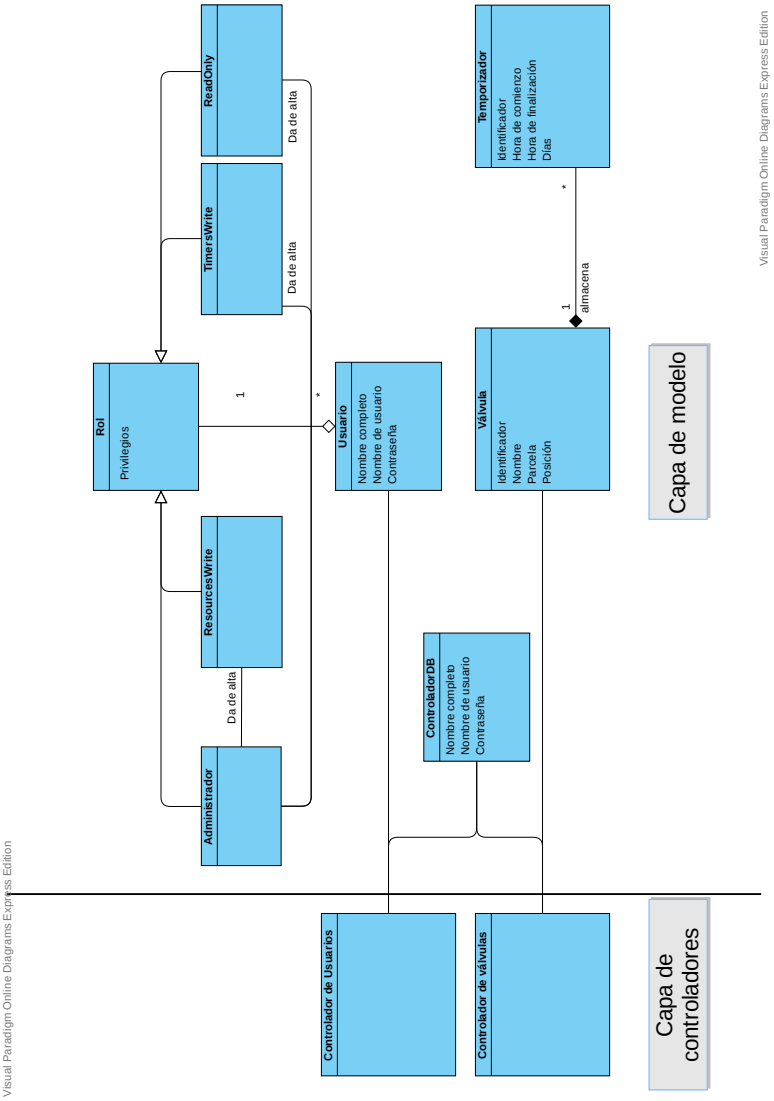

FIGURA B.1: Diagrama de modelo del diseño

B.2. Casos de uso de diseño del sistema

B.2.1. Definición de actores

Además de los actores identificados en la fase de análisis (ver sección A.3.1), durante el diseño se han identificado los siguientes actores adicionales:

- Arduino: Actor que representa al software instalado en cada placa arduino, que se encarga de enviar, durante su inicialización, la petición para la creación de una válvula en la aplicación de control.

B.2.2. Diagrama de casos de uso

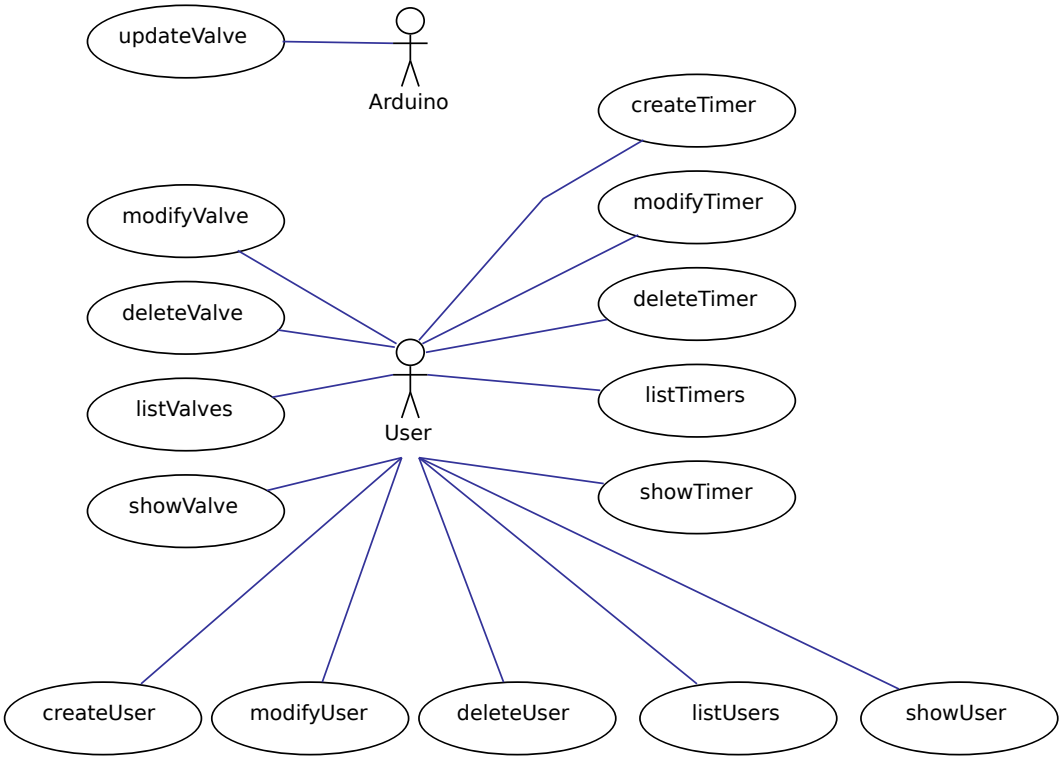

FIGURA B.2: Diagrama de casos de uso del sistema

B.2.3. Diagramas de secuencia de diseño del sistema

Creación de usuario

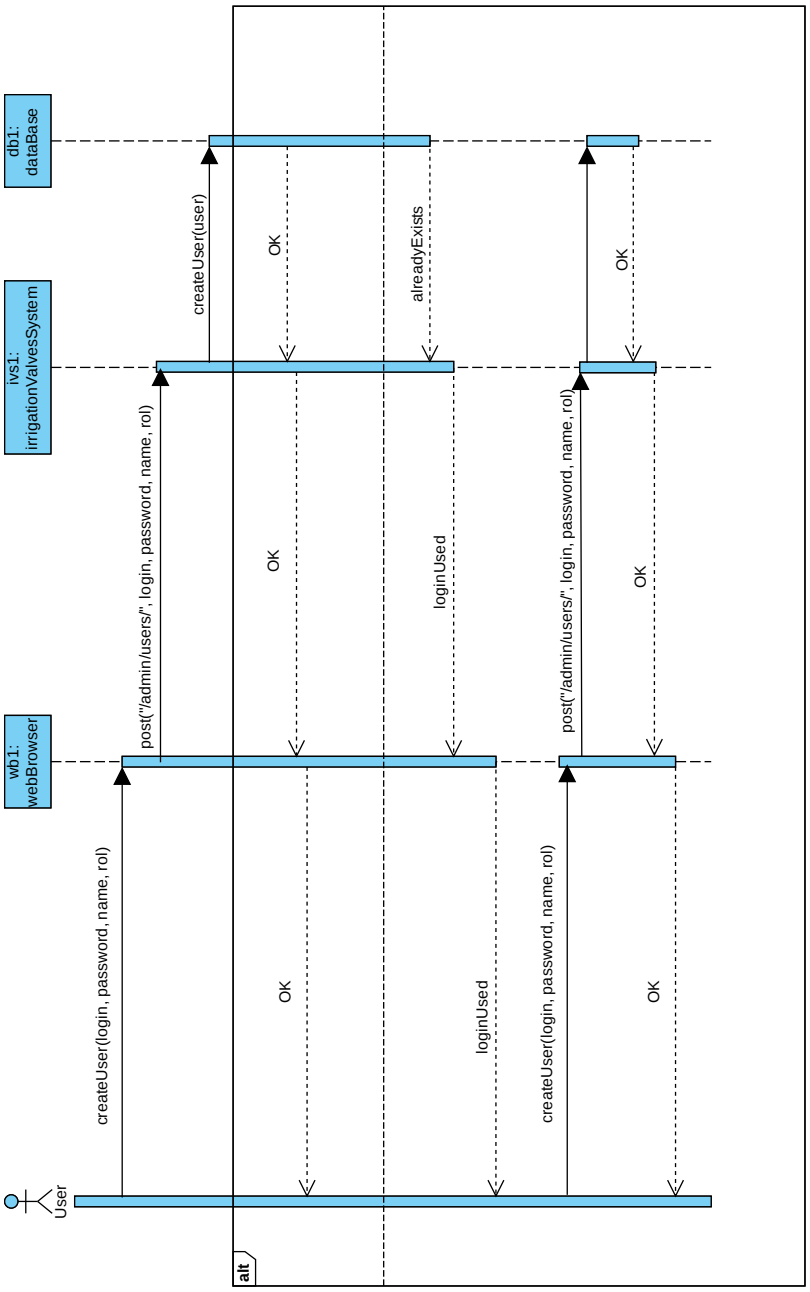

FIGURA B.3: Caso de uso de creación de los usuarios

Listado de usuarios de la aplicación

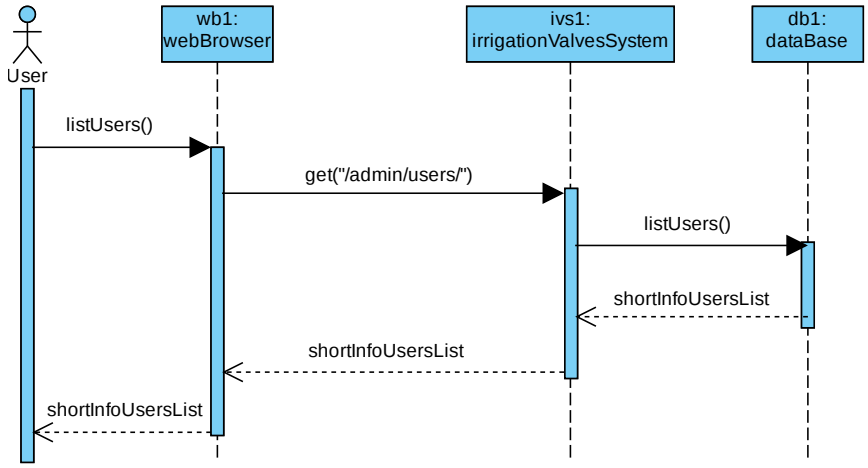

FIGURA B.4: Caso de uso de listado de los usuarios

Visualización de usuario

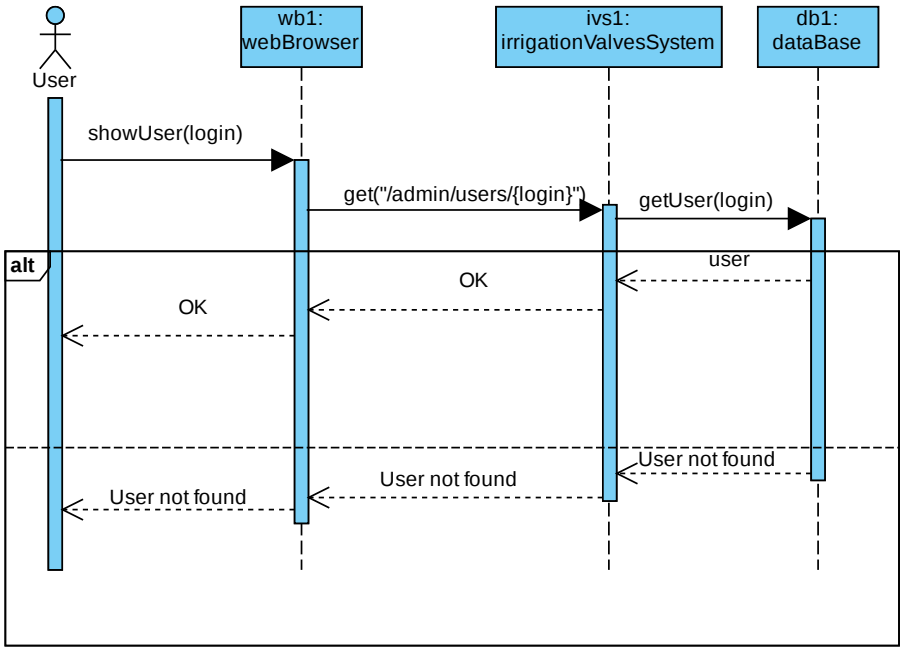

FIGURA B.5: Caso de uso de visualización de un usuario

Modificación de usuario

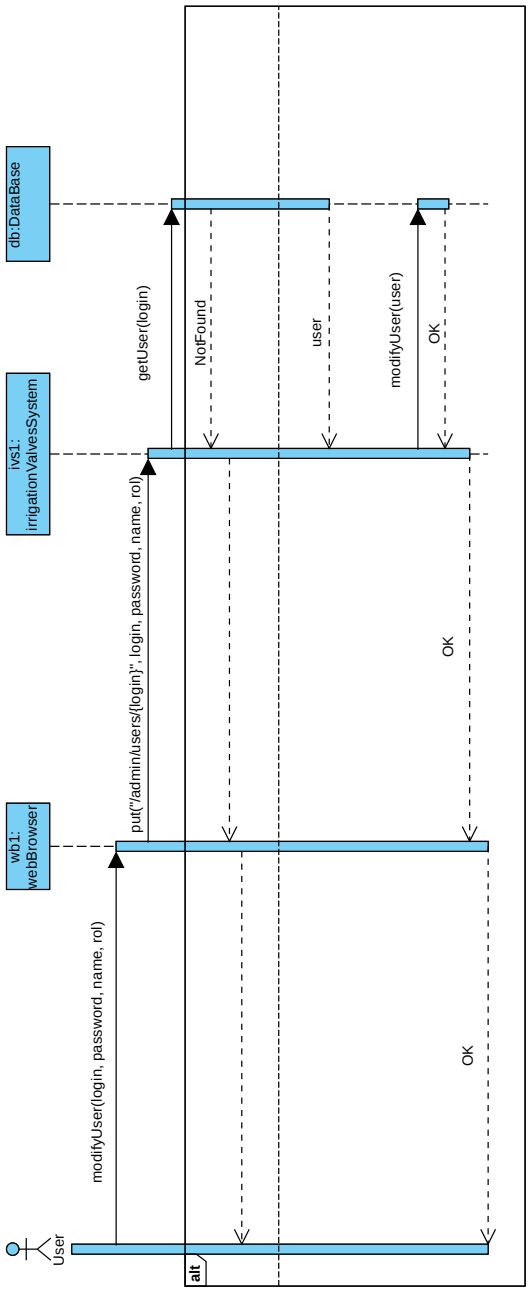

FIGURA B.6: Caso de uso de modificación de un usuario

Borrado de usuario

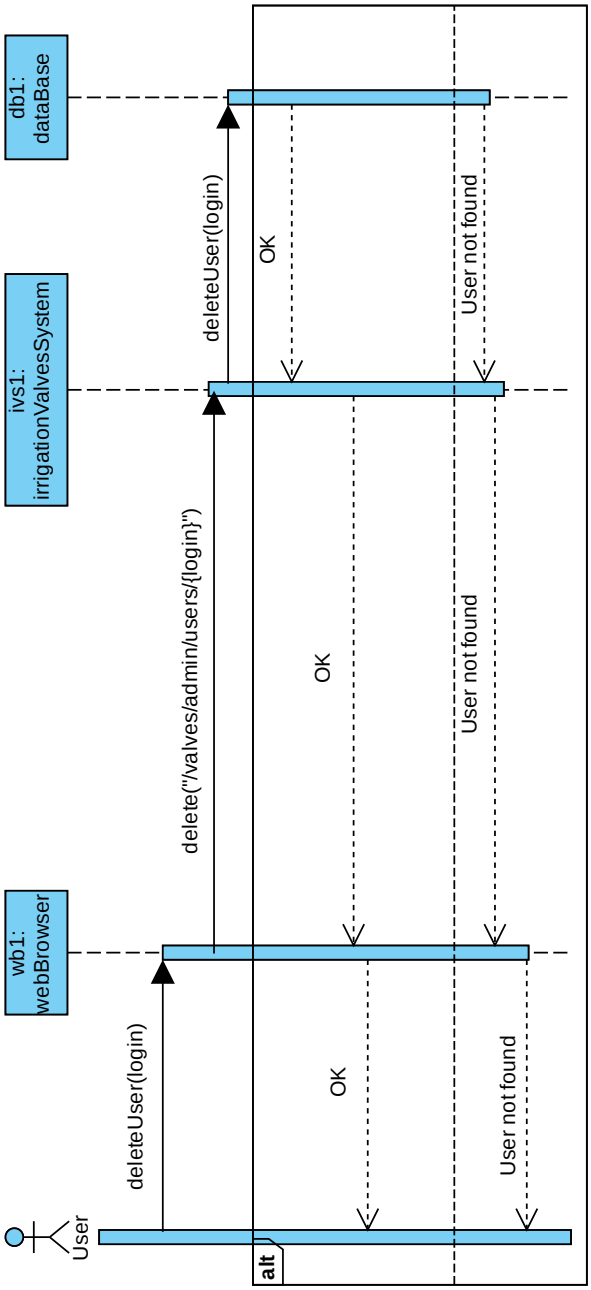

FIGURA B.7: Caso de uso de borrado de un usuario

Creación de válvula

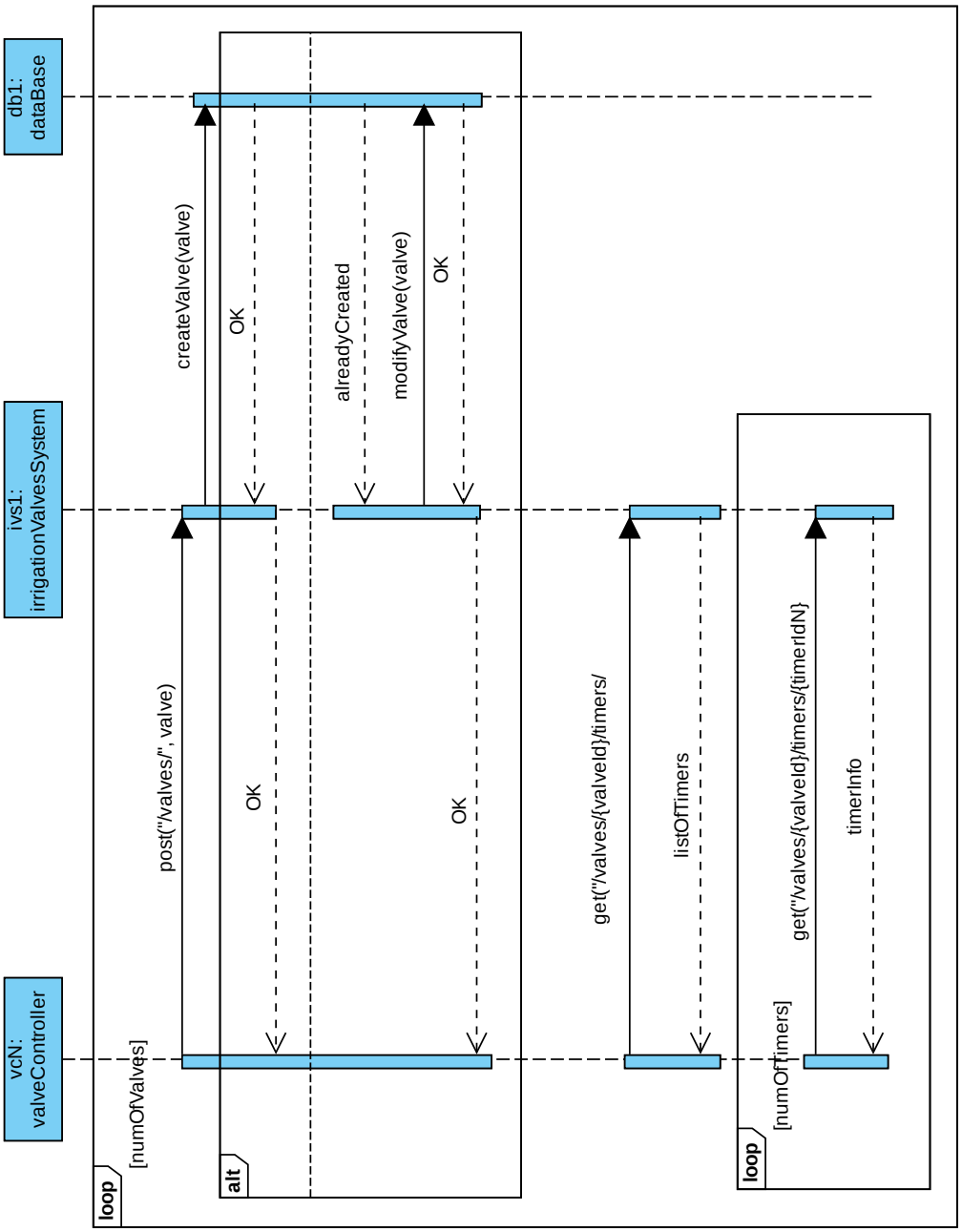

FIGURA B.8: Caso de uso de creación de la válvula

Listado de válvulas

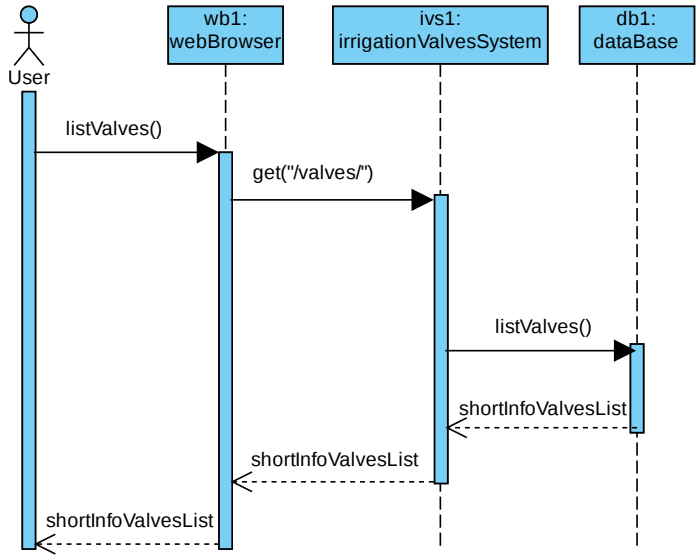

FIGURA B.9: Caso de uso de listado de las válvulas

Visualización de la válvula

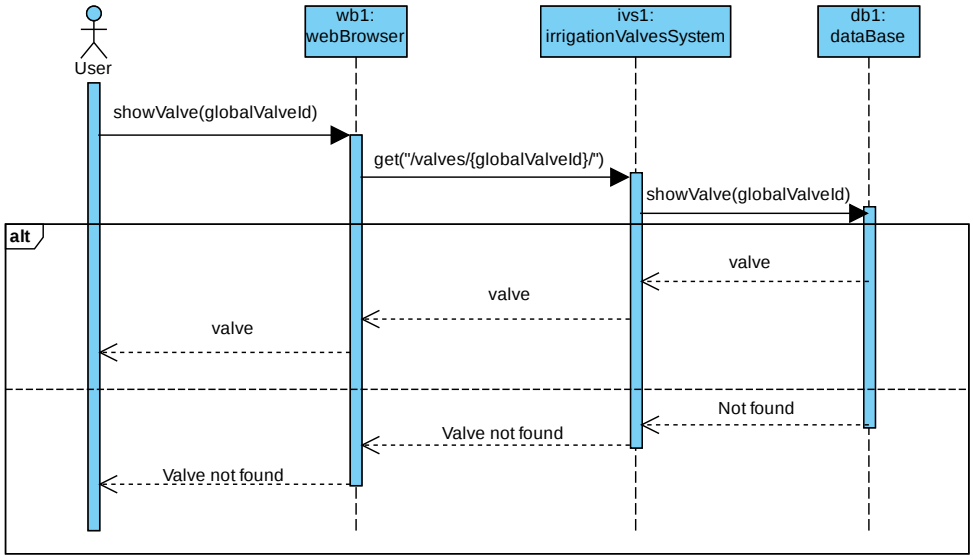

FIGURA B.10: Caso de uso de visualización de la válvula

Modificación de válvula

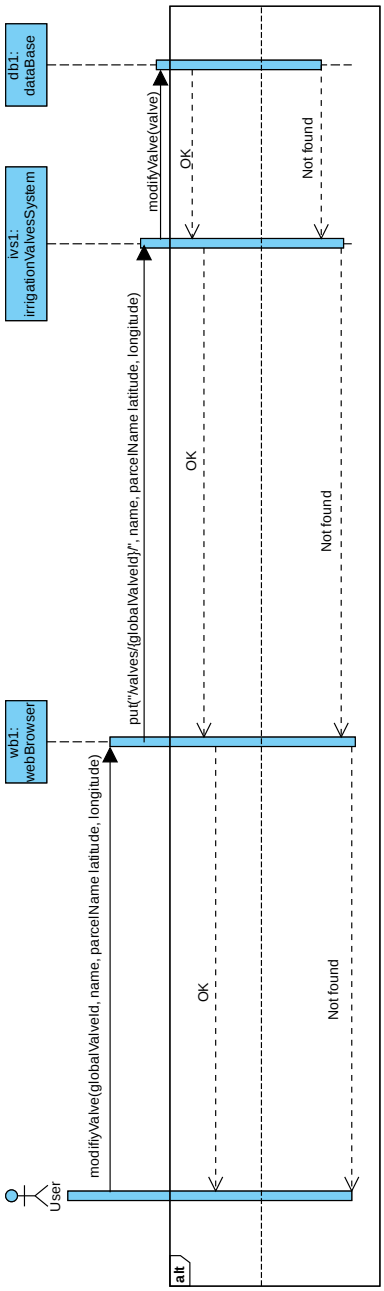

FIGURA B.11: Caso de uso de modificación de válvula

Apertura y cierre de válvula

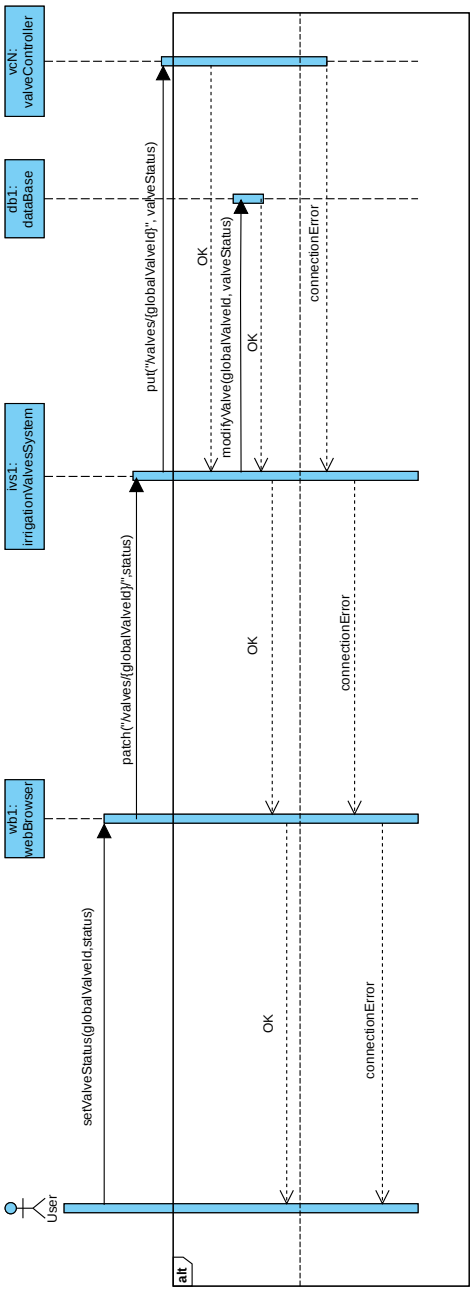

FIGURA B.12: Caso de uso de apertura y cierre de válvula

Borrado de válvula

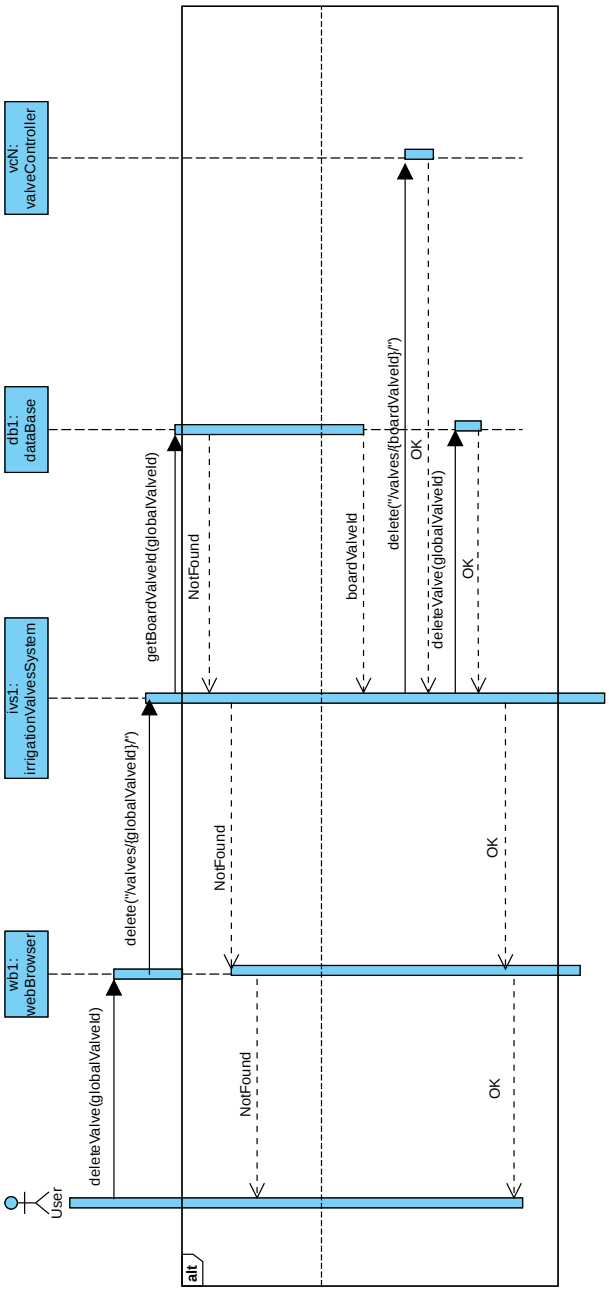

FIGURA B.13: Caso de uso de borrado de válvula

Vencimiento del temporizador

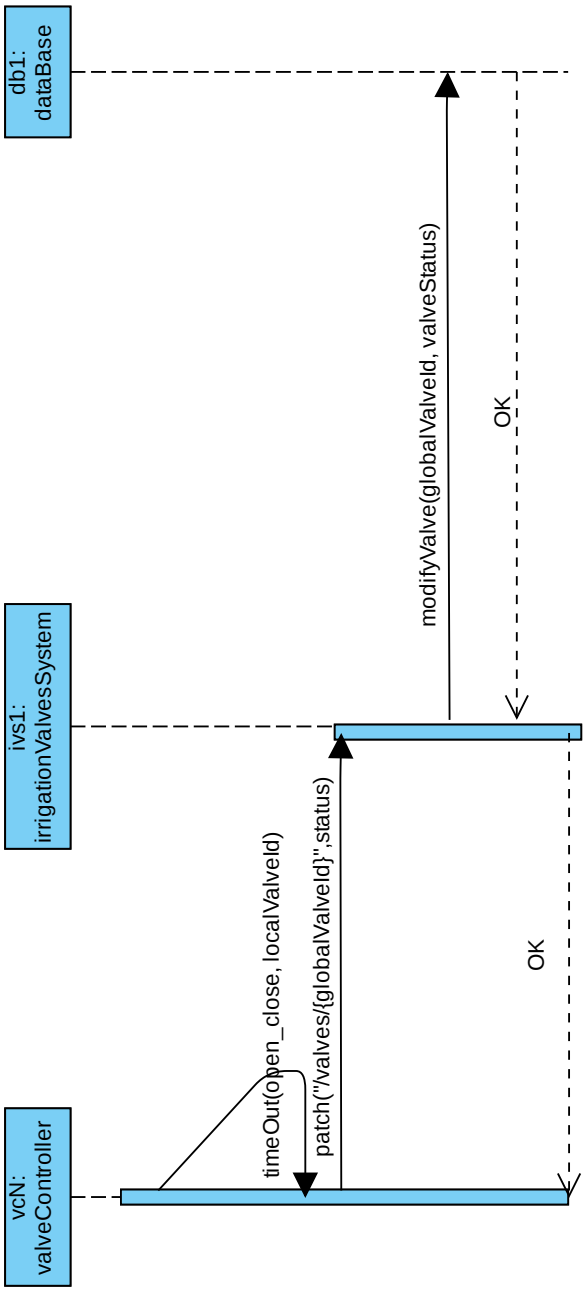

FIGURA B.14: Caso de uso de vencimiento del temporizador

Creación de temporizador

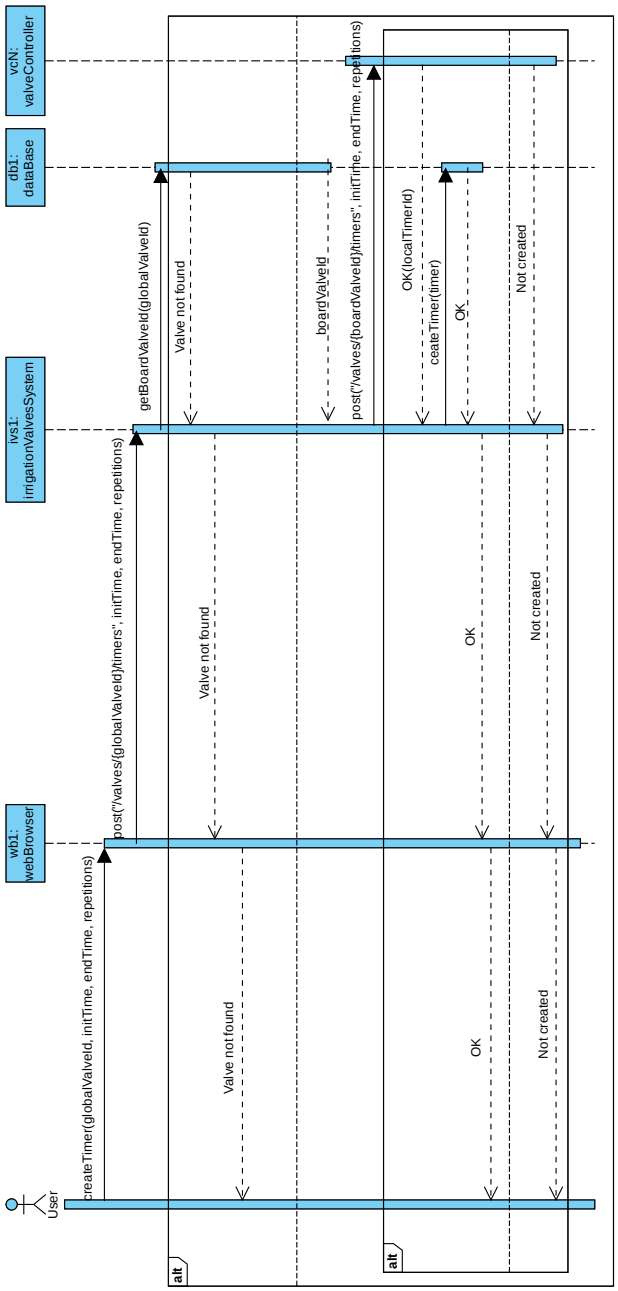

FIGURA B.15: Caso de uso de creación de un temporizador

Listado de temporizadores

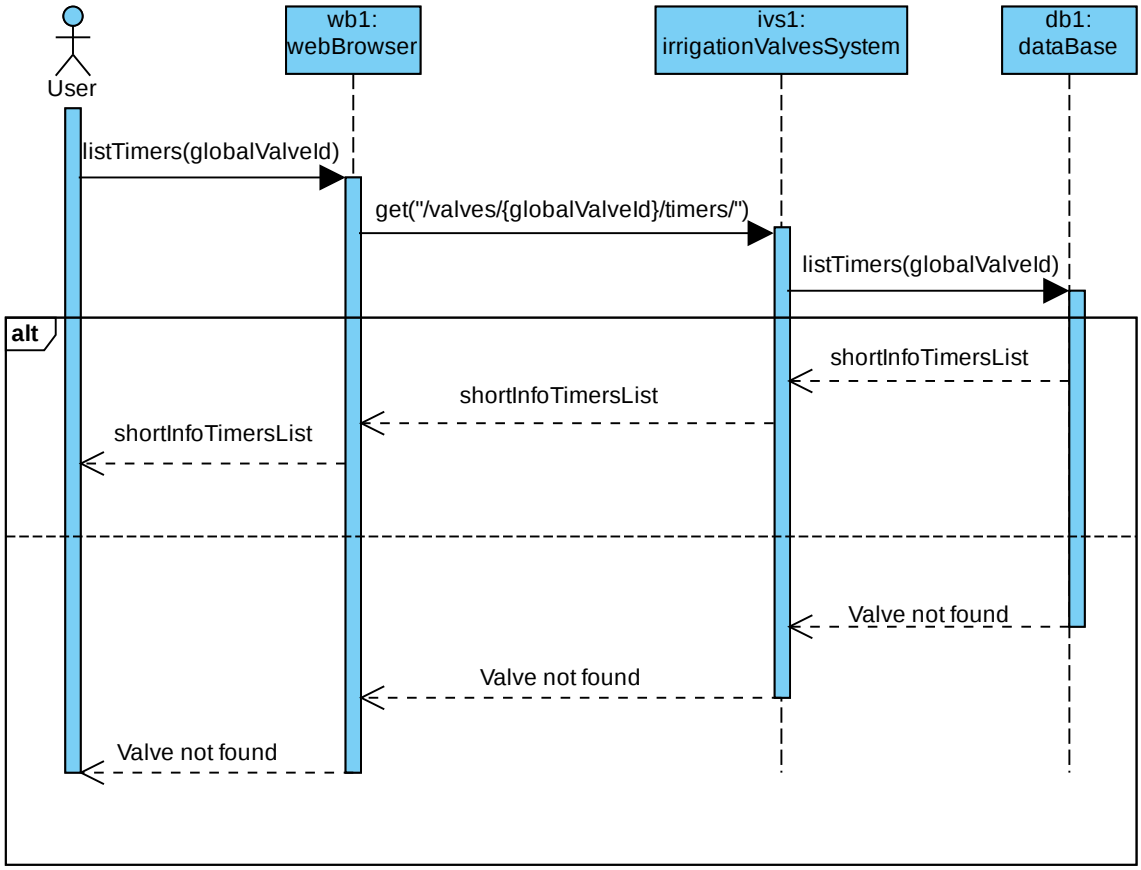

FIGURA B.16: Caso de uso de listado de temporizadores

Visualización de temporizador

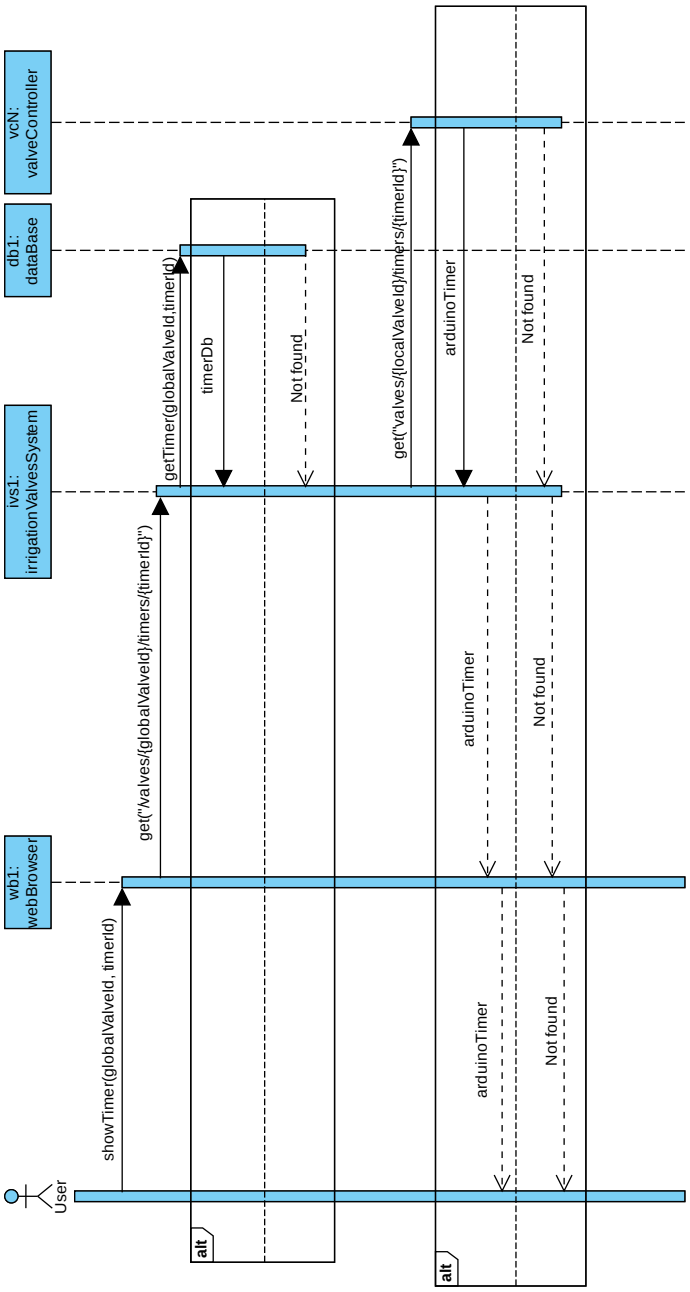

FIGURA B.17: Caso de uso de visualización del temporizador

Modificación de temporizador

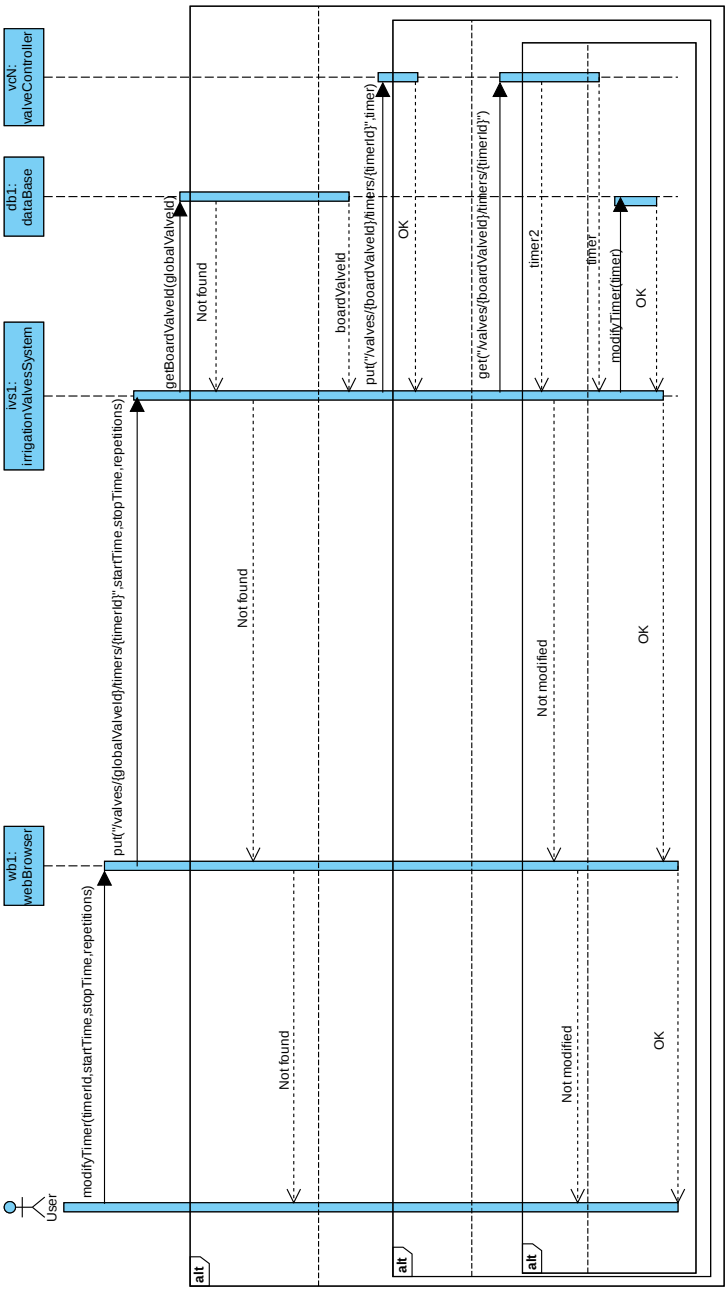

FIGURA B.18: Caso de uso de modificación de temporizador

Borrado de temporizador

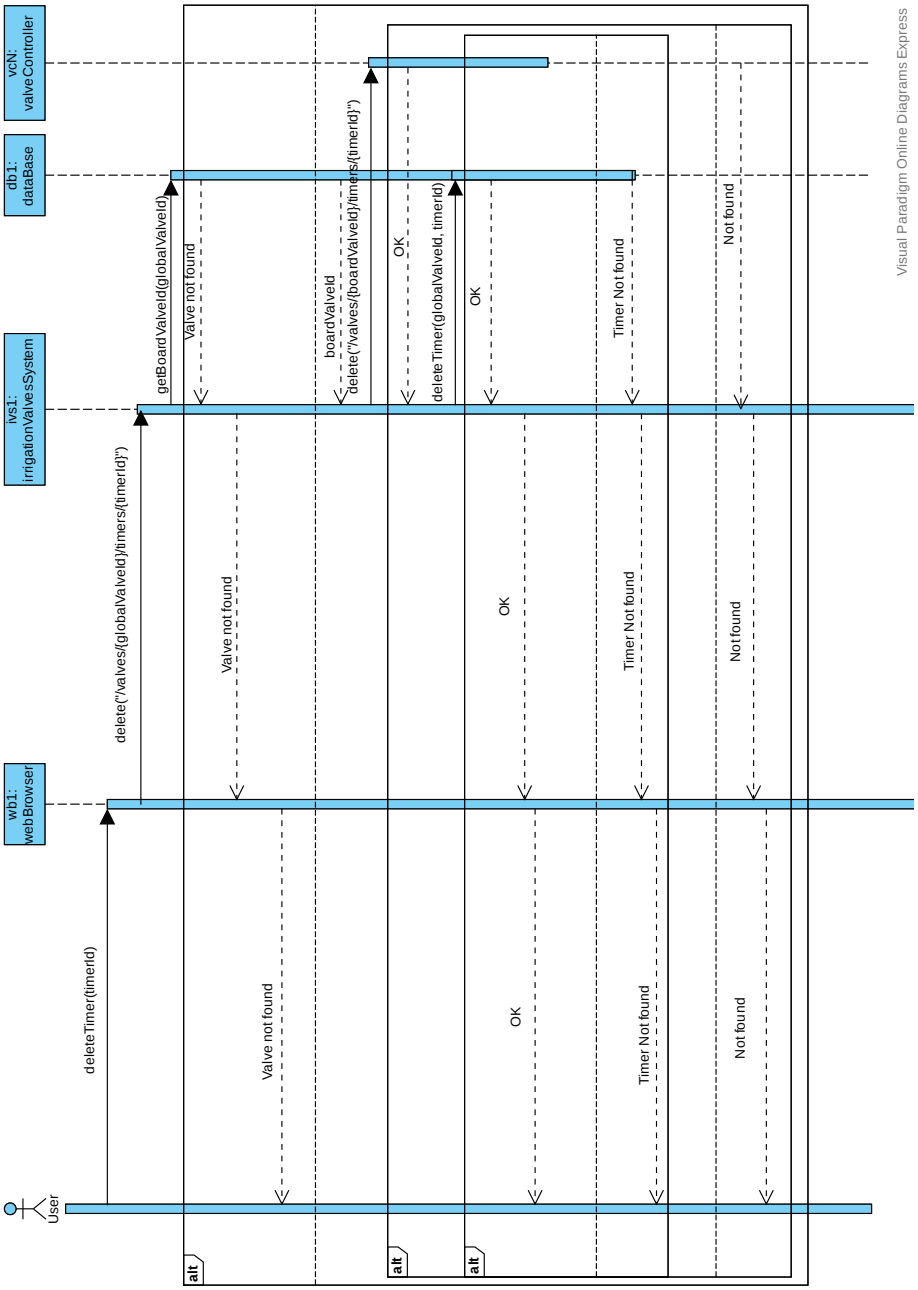

FIGURA B.19: Caso de uso de borrado de temporizador

B.3. Servicio web *Smartgranja*

B.3.1. Modelo de recursos del servicio web

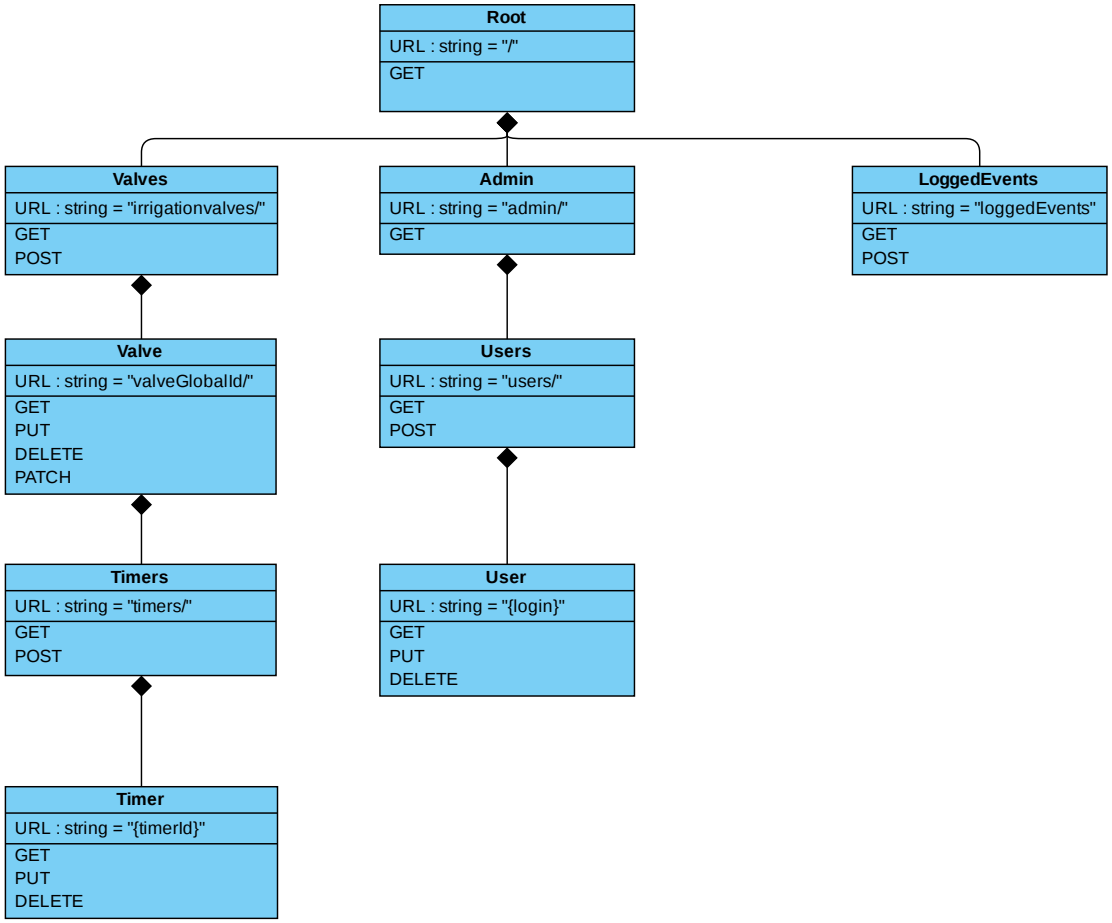

FIGURA B.20: Diagrama de recursos de la aplicación java

B.3.2. Autorización de usuarios

	Recursos		
Roles	**Person(s)**	**Valve(s)**	**Timer(s)**
Admin	R/W	R/W	R/W
ResourcesWrite	—	R/W	R/W
TimersWrite	—	R	R/W
ReadOnly	—	R	R

CUADRO B.1: Roles de usuarios y operaciones permitidas sobre recursos (R/W: lectura y escritura; R: solo lectura; —: sin permisos)

B.4. Aplicación Arduino

B.4.1. Modelo de recursos del servicio web

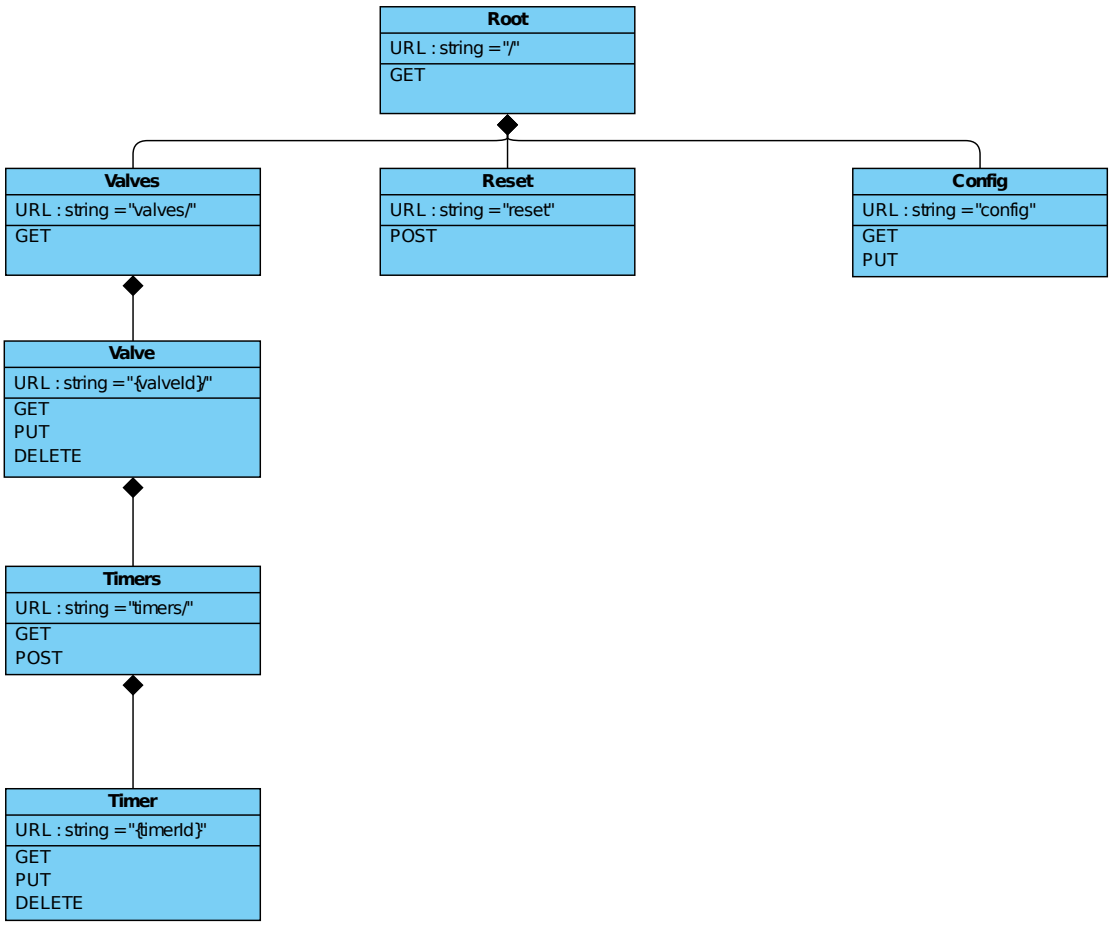

FIGURA B.21: Diagrama de recursos de la aplicación Arduino

Apéndice C

Documentos de Pruebas

C.1. Conjuntos de pruebas

C.1.1. Listado de usuarios

Aspecto	**Especificación**
Nombre	Listado de los usuarios existentes
Tipo de prueba	Integración/Aceptación
Criticidad	Media
Precondiciones	Existe un usuario administrador y se ha ingresado al sistema con su nombre de usuario.
Escenario	El usuario accede a las opciones de gestión y, a la gestión de usuarios.
Postcondiciones	Se muestran los usuarios existentes con información abreviada.

Cuadro C.1: Listado de usuarios

Aspecto	Especificación
Nombre	Listado de los usuarios existentes sin privilegios suficientes
Tipo de prueba	Aceptación
Criticidad	Alta
Precondiciones	Existe un usuario con el rol asignado del ejemplo y se ha ingresado al sistema con su nombre de usuario.
Escenario	El usuario solicita el listado de usuarios por URL.
Postcondiciones	Se muestra un mensaje de error y se redirige a la pantalla principal de la aplicación.
Ejemplos (Roles)	ResourcesWrite TimersWrite ReadOnly

Cuadro C.2: Listado de usuarios sin privilegios suficientes

C.1.2. Creación de usuarios

Aspecto	Especificación
Nombre	Creación de usuarios
Tipo de prueba	Integración/Aceptación
Criticidad	Media
Precondiciones	Existe un usuario administrador y se ha ingresado al sistema con su «nombre de usuario»
Escenario	El usuario accede a las opciones de administración, accede a la gestión de usuarios, solicita el formulario para la creación de un usuario y solicita, finalmente, la creación del usuario con los datos y, en cada ejecución, uno de los roles.
Postcondiciones	Se ha creado el usuario Se muestra la información del usuario
Ejemplos (Roles)	ResourcesWrite Admin TimersWrite ReadOnly

Cuadro C.3: Creación de usuarios

Aspecto	Especificación
Nombre	Crea un usuario con parámetros erróneos
Tipo de prueba	Integración/Aceptación
Criticidad	Media
Precondiciones	Existe un usuario administrador y se ha ingresado al sistema con su «nombre de usuario»
Escenario	El usuario accede a las opciones de administración, accede a la gestión de usuarios, solicita el formulario para la creación de un usuario y solicita, finalmente, la creación del usuario con los datos erróneos.
Postcondiciones	Se muestra un mensaje indicando el error

Cuadro C.4: Creación de usuarios con argumentos erróneos

Aspecto	**Especificación**
Nombre	Acceder a la creación de usuario desde un usuario sin privilegios suficientes
Tipo de prueba	Aceptación
Criticidad	Alta
Precondiciones	Existe un usuario con el rol asignado del ejemplo y se ha ingresado al sistema con su nombre de usuario.
Escenario	El usuario solicita el formulario para la creación de un usuario por URL.
Postcondiciones	Se muestra un mensaje de error y se redirige a la pantalla principal de la aplicación.
Ejemplos (Roles)	ResourcesWrite TimersWrite ReadOnly

Cuadro C.5: Creación de usuarios sin privilegios suficientes

C.1.3. Visualización de usuarios

Aspecto	Especificación
Nombre	Visualización de usuarios
Tipo de prueba	Integración/Aceptación
Criticidad	Media
Precondiciones	Existe un usuario administrador y se ha ingresado al sistema con su «nombre de usuario»
Escenario	El usuario accede a las opciones de administración, accede a la gestión de usuarios y solicita la información de un usuario.
Postcondiciones	Se muestra la información del usuario

Cuadro C.6: Visualización de usuarios

Aspecto	Especificación
Nombre	Acceder a la visualización de usuarios desde un usuario sin privilegios suficientes
Tipo de prueba	Aceptación
Criticidad	Alta
Precondiciones	Existe un usuario con el rol asignado del ejemplo y se ha ingresado al sistema con su nombre de usuario.
Escenario	El usuario solicita la información de un usuario por URL.
Postcondiciones	Se muestra un mensaje de error y se redirige a la pantalla principal de la aplicación.
Ejemplos (Roles)	ResourcesWrite TimersWrite ReadOnly

Cuadro C.7: Visualización de usuarios sin privilegios suficientes

Aspecto	Especificación
Nombre	Acceder a la visualización de un usuario inexistente
Tipo de prueba	Integración/Aceptación
Criticidad	Media
Precondiciones	Se ha ingresado al sistema
Escenario	El usuario solicita la información de un usuario inexistente por URL.
Postcondiciones	Se muestra un mensaje de error y se redirige al listado de usuarios.

Cuadro C.8: Visualización de un usuario inexistente

C.1.4. Modificación de usuarios

Aspecto	Especificación
Nombre	Modificación de usuarios
Tipo de prueba	Integración/Aceptación
Criticidad	Media
Precondiciones	Existe un usuario administrador y se ha ingresado al sistema con su «nombre de usuario»
Escenario	El usuario accede a las opciones de administración, accede a la gestión de usuarios, solicita el formulario para la modificación de un usuario y solicita, finalmente, la modificación del usuario con los datos.
Postcondiciones	Se ha modificado el usuario Se muestra la información del usuario

Cuadro C.9: Modificación de usuarios

Aspecto	Especificación
Nombre	Modifica un usuario con parámetros erróneos
Tipo de prueba	Integración/Aceptación
Criticidad	Media
Precondiciones	Existe un usuario administrador y se ha ingresado al sistema con su «nombre de usuario»
Escenario	El usuario accede a las opciones de administración, accede a la gestión de usuarios, solicita el formulario para la modificación de un usuario y solicita, finalmente, la modificación del usuario con los datos erróneos.
Postcondiciones	Se muestra un mensaje indicando el error

Cuadro C.10: Modificación de usuarios con argumentos erróneos

Aspecto	Especificación
Nombre	Acceder a la modificación de usuarios desde un usuario sin privilegios suficientes
Tipo de prueba	Aceptación
Criticidad	Alta
Precondiciones	Existe un usuario con el rol asignado del ejemplo y se ha ingresado al sistema con su nombre de usuario.
Escenario	El usuario solicita el formulario para la modificación de un usuario por URL.
Postcondiciones	Se muestra un mensaje de error y se redirige a la pantalla principal de la aplicación.
Ejemplos (Roles)	ResourcesWrite TimersWrite ReadOnly

Cuadro C.11: Modificación de usuarios sin privilegios suficientes

C.1.5. Borrado de usuarios

Aspecto	Especificación
Nombre	Borrado de usuarios
Tipo de prueba	Integración/Aceptación
Criticidad	Media
Precondiciones	Existe un usuario administrador y se ha ingresado al sistema con su «nombre de usuario»
Escenario	El usuario accede a las opciones de administración, accede a la gestión de usuarios y solicita el borrado de un usuario.
Postcondiciones	Se borra el usuario Se muestra el listado de usuarios

Cuadro C.12: Borrado de usuarios

Aspecto	Especificación
Nombre	Acceder al borrado de usuarios desde un usuario sin privilegios suficientes
Tipo de prueba	Aceptación
Criticidad	Alta
Precondiciones	Existe un usuario con el rol asignado del ejemplo y se ha ingresado al sistema con su nombre de usuario.
Escenario	El usuario solicita el formulario de borrado de un usuario por URL.
Postcondiciones	Se muestra un mensaje de error y se redirige a la pantalla principal de la aplicación.
Ejemplos (Roles)	ResourcesWrite TimersWrite ReadOnly

Cuadro C.13: Borrado de usuarios sin privilegios suficientes

C.1.6. Listado de válvulas

Aspecto	Especificación
Nombre	Listado de las válvulas existentes
Tipo de prueba	Integración/Aceptación
Criticidad	Media
Precondiciones	Existe un usuario con el rol del ejemplo y se ha ingresado al sistema con su nombre de usuario.
Escenario	El usuario accede a las opciones de gestión del sistema de riego y, al listado de válvulas.
Postcondiciones	Se muestran las válvulas existentes con información abreviada.
Ejemplos (Roles)	ResourcesWrite Admin TimersWrite ReadOnly

Cuadro C.14: Listado de válvulas

C.1.7. Visualización de válvulas

Aspecto	Especificación
Nombre	Visualización de válvulas
Tipo de prueba	Integración/Aceptación
Criticidad	Media
Precondiciones	Existe un usuario con el rol del ejemplo y se ha ingresado al sistema con su «nombre de usuario»
Escenario	El usuario accede a las opciones de gestión del sistema de riego, al listado de válvulas y solicita una válvula.
Postcondiciones	Se muestra la información de la válvula
Ejemplos (Roles)	ResourcesWrite Admin TimersWrite ReadOnly

Cuadro C.15: Visualización de válvulas

Aspecto	Especificación
Nombre	Acceder a la visualización de válvulas introduciendo un identificador incorrecto
Tipo de prueba	Integración/Aceptación
Criticidad	Media
Precondiciones	Se ha ingresado al sistema
Escenario	El usuario accede a las opciones de gestión del sistema de riego, al listado de válvulas y solicita una válvula inexistente.
Postcondiciones	Se muestra un mensaje de error Se redirige al listado de válvulas

Cuadro C.16: Visualización de válvulas incorrecta

C.1.8. Modificación de válvulas

Aspecto	Especificación
Nombre	Modificación de válvulas
Tipo de prueba	Integración/Aceptación
Criticidad	Media
Precondiciones	Existe un usuario con el rol del ejemplo y ha ingresado al sistema con su nombre de usuario.
Escenario	El usuario accede a las opciones de gestión del sistema de riego, al listado de válvulas, a la modificación de una válvula, introduce los datos y solicita la modificación.
Postcondiciones	Se modifica la válvula Se muestran la información de la válvula modificada.
Ejemplos (Roles)	ResourcesWrite Admin

Cuadro C.17: Modificación de válvulas

Aspecto	Especificación
Nombre	Modificación de válvulas con información errónea
Tipo de prueba	Integración/Aceptación
Criticidad	Media
Precondiciones	Existe un usuario con el rol ResourcesWrite ó Admin y ha ingresado al sistema con su nombre de usuario.
Escenario	El usuario accede a las opciones de gestión del sistema de riego, al listado de válvulas, a la modificación de una válvula, introduce los datos de forma incorrecta y solicita la modificación.
Postcondiciones	Se muestra un mensaje de error y se redirige a la información de la válvula

Cuadro C.18: Modificación de válvulas con información errónea

Aspecto	Especificación
Nombre	Acceder a la modificación de válvulas, a través de URL, sin privilegios suficientes
Tipo de prueba	Aceptación
Criticidad	Alta
Precondiciones	Existe un usuario con el rol del ejemplo y ha ingresado al sistema con su nombre de usuario.
Escenario	El usuario accede a la modificación de una válvula existente, a través de la URL.
Postcondiciones	Se muestra un mensaje de error y se redirige a la información de la válvula
Ejemplos (Roles)	TimersWrite ReadOnly

Cuadro C.19: Modificación de válvulas sin privilegios suficientes

C.1.9. Borrado de válvulas

Aspecto	Especificación
Nombre	Borrado de válvulas
Tipo de prueba	Integración
Criticidad	Media
Precondiciones	Existe un usuario con el rol del ejemplo y ha ingresado al sistema con su nombre de usuario.
Escenario	El usuario accede a las opciones de gestión del sistema de riego, al listado de válvulas, al formulario de borrado de una válvula y solicita el borrado.
Postcondiciones	Se elimina la válvula Se muestran la lista de válvulas existentes.
Ejemplos (Roles)	ResourcesWrite Admin

Cuadro C.20: Borrado de válvulas

Aspecto	Especificación
Nombre	Borrado de válvulas (Aceptación)
Tipo de prueba	Aceptación
Criticidad	Alta
Precondiciones	Existe un usuario con el rol del ejemplo y ha ingresado al sistema con su nombre de usuario.
Escenario	El usuario accede a las opciones de gestión del sistema de riego, al listado de válvulas, al formulario de borrado de una válvula y solicita el borrado.
Postcondiciones	Se elimina la válvula Se eliminan todos los temporizadores disponibles Se cumplen todas las postcondiciones de la prueba C.34 referidas al controlador para cada uno de los temporizadores contenidos en la válvula Se muestran la lista de válvulas existentes.
Ejemplos (Roles)	ResourcesWrite Admin

Cuadro C.21: Borrado de válvulas (Aceptación)

Aspecto	Especificación
Nombre	Acceder al borrado de válvulas, a través de URL, sin privilegios suficientes
Tipo de prueba	Aceptación
Criticidad	Alta
Precondiciones	Existe un usuario con el rol del ejemplo y ha ingresado al sistema con su nombre de usuario.
Escenario	El usuario accede al borrado de una válvula existente, a través de la URL.
Postcondiciones	Se muestra un mensaje de error y se redirige a la información de la válvula
Ejemplos (Roles)	TimersWrite ReadOnly

Cuadro C.22: Borrado de válvulas sin privilegios suficientes

C.1.10. Listado de temporizadores

Aspecto	Especificación
Nombre	Listado de los temporizadores existentes
Tipo de prueba	Integración/Aceptación
Criticidad	Media
Precondiciones	Existe un usuario con el rol del ejemplo y ha ingresado al sistema con su nombre de usuario.
Escenario	El usuario accede a las opciones de gestión del sistema de riego, al listado de válvulas, a una válvula concreta y, finalmente, al listado de sus temporizadores.
Postcondiciones	Se muestran los temporizadores existentes con información abreviada.
Ejemplos (Roles)	ResourcesWrite Admin TimersWrite ReadOnly

Cuadro C.23: Listado de temporizadores

C.1.11. Creación de temporizadores

Aspecto	Especificación
Nombre	Creación de temporizadores
Tipo de prueba	Integración/Aceptación
Criticidad	Media
Precondiciones	Existe un usuario con el rol del ejemplo y se ha ingresado al sistema con su nombre de usuario.
Escenario	El usuario accede a las opciones de gestión del sistema de riego, al listado de válvulas, a una válvula concreta, al listado de sus temporizadores, selecciona la creación de un nuevo temporizador, introduce los datos y solicita la creación.
Postcondiciones	Se muestran la información del temporizador creado.
Ejemplos (Roles)	ResourcesWrite Admin TimersWrite

Cuadro C.24: Creación de temporizadores

Aspecto	Especificación
Nombre	Acceder a la creación de temporizadores, a través de URL, sin privilegios suficientes
Tipo de prueba	Aceptación
Criticidad	Alta
Precondiciones	Existe un usuario con rol readOnly y se ha ingresado al sistema con su nombre de usuario.
Escenario	El usuario accede a la creación de un nuevo temporizador a través de la URL.
Postcondiciones	Se muestra un mensaje de error y se redirige a la pantalla con el listado de los temporizadores.

Cuadro C.25: Creación de temporizadores sin privilegios suficientes

Aspecto	Especificación
Nombre	Crear temporizador que no se deba activar antes de los previamente existentes
Tipo de prueba	Aceptación
Criticidad	Media
Precondiciones	Existe un usuario con un rol que permita crear temporizadores (ResourcesWrite, TimersWrite ó Admin) y se ha ingresado al sistema con su nombre de usuario. Tener creada una válvula al menos Tener creados temporizadores en esa válvula
Escenario	El usuario crea un temporizador (el proceso es el utilizado en el escenario C.24) y, crea un temporizador con hora posterior a los ya existentes en el día actual.
Postcondiciones	Después de activar los temporizadores previos, el arduino debe fijar la alarma a esa hora Cuando llegue su hora de inicio debería activarse la válvula Debe mostrarse la información del temporizador

Cuadro C.26: Creación de temporizadores antes de los existentes

Aspecto	**Especificación**
Nombre	Crear temporizador marcando días diferentes al actual y otro posterior (en hora) pero en el día actual
Tipo de prueba	Aceptación
Criticidad	Media
Precondiciones	Existe un usuario con un rol que permita crear temporizadores (ResourcesWrite, TimersWrite ó Admin) y se ha ingresado al sistema con su nombre de usuario. Tener creada una válvula al menos
Escenario	El usuario crea un temporizador (el proceso es el utilizado en el escenario C.24) y, crea un temporizador en un día diferente al actual. Posteriormente, se crea otro temporizador con una hora posterior al previamente creado y con el día actual.
Postcondiciones	Después de activar los temporizadores previos, el arduino debe fijar la alarma a esa hora Cuando llegue su hora de inicio debería activarse la válvula Cuando llegue su hora de finalización debería desactivarse la válvula Debe mostrarse la información del temporizador

CUADRO C.27: Prueba acerca del funcionamiento de los días marcados en la creación de un temporizador

C.1.12. Visualización de temporizadores

Aspecto	Especificación
Nombre	Visualización de temporizadores
Tipo de prueba	Integración/Aceptación
Criticidad	Media
Precondiciones	Existe un usuario con el rol del ejemplo y se ha ingresado al sistema con su nombre de usuario.
Escenario	El usuario accede a las opciones de gestión del sistema de riego, al listado de válvulas, a una válvula concreta, al listado de sus temporizadores y selecciona un temporizador para su visualización
Postcondiciones	Se muestra la información del temporizador solicitado.
Ejemplos (Roles)	ResourcesWrite Admin ReadOnly TimersWrite

Cuadro C.28: Visualización de temporizadores

Aspecto	Especificación
Nombre	Visualización de temporizadores introduciendo un identificador incorrecto
Tipo de prueba	Integración/Aceptación
Criticidad	Media
Precondiciones	Se ha ingresado al sistema
Escenario	El usuario introduce la URL de un temporizador inexistente
Postcondiciones	Se muestra un mensaje de error Se redirige al listado de temporizadores

Cuadro C.29: Visualización de temporizadores introduciendo un identificador incorrecto

C.1.13. Modificación de temporizadores

Aspecto	Especificación
Nombre	Modificación de temporizadores
Tipo de prueba	Integración
Criticidad	Media
Precondiciones	Existe un usuario con el rol del ejemplo y se ha ingresado al sistema con su nombre de usuario.
Escenario	El usuario accede a las opciones de gestión del sistema de riego, al listado de válvulas, a una válvula concreta, al listado de sus temporizadores, selecciona la modificación de un temporizador existente, introduce los datos y solicita la modificación.
Postcondiciones	Se muestran la información del temporizador modificado.
Ejemplos (Roles)	ResourcesWrite Admin TimersWrite

Cuadro C.30: Modificación de temporizadores

Aspecto	Especificación
Nombre	Modificación de temporizadores (Aceptación)
Tipo de prueba	Aceptación
Criticidad	Alta
Precondiciones	Existe un usuario con el rol del ejemplo y se ha ingresado al sistema con su nombre de usuario.
Escenario	El usuario accede a las opciones de gestión del sistema de riego, al listado de válvulas, a una válvula concreta, al listado de sus temporizadores, selecciona la modificación de un temporizador existente, introduce los datos y solicita la modificación.
Postcondiciones	Después de activar los temporizadores previos, el arduino debe fijar la alarma a esa hora Cuando llegue su hora de inicio debería activarse la válvula Se muestran la información del temporizador modificado.
Ejemplos (Roles)	ResourcesWrite Admin TimersWrite

Cuadro C.31: Modificación de temporizadores (aceptación)

Aspecto	Especificación
Nombre	Acceder a la modificación de temporizadores, a través de URL, sin privilegios suficientes
Tipo de prueba	Aceptación
Criticidad	Alta
Precondiciones	Existe un usuario con rol readOnly y se ha ingresado al sistema con su nombre de usuario.
Escenario	El usuario accede a la modificación de un temporizador existente a través de la URL.
Postcondiciones	Se muestra un mensaje de error y se redirige a la pantalla con la información del temporizador.

Cuadro C.32: Modificación de temporizadores sin privilegios suficientes

C.1.14. Borrado de temporizadores

Aspecto	Especificación
Nombre	Borrado de los temporizadores
Tipo de prueba	Integración
Criticidad	Media
Precondiciones	Existe un usuario con el rol del ejemplo y se ha ingresado al sistema con su nombre de usuario.
Escenario	El usuario accede a las opciones de gestión del sistema de riego, al listado de válvulas, a una válvula concreta, al listado de sus temporizadores, selecciona el borrado de un temporizador existente.
Postcondiciones	Se muestra el listado de temporizadores disponibles.
Ejemplos (Roles)	ResourcesWrite Admin TimersWrite

Cuadro C.33: Borrado de temporizadores

Aspecto	Especificación
Nombre	Borrado de los temporizadores (Aceptación)
Tipo de prueba	Aceptación
Criticidad	Alta
Precondiciones	Existe un usuario con el rol del ejemplo y se ha ingresado al sistema con su nombre de usuario.
Escenario	El usuario accede a las opciones de gestión del sistema de riego, al listado de válvulas, a una válvula concreta, al listado de sus temporizadores, selecciona el borraod de un temporizador existente.
Postcondiciones	Se desactiva la alarma en el controlador en el caso de que la alarma activada corresponda al temporizador Se muestra el listado de temporizadores disponibles.
Ejemplos (Roles)	ResourcesWrite Admin TimersWrite

Cuadro C.34: Borrado de temporizadores (Aceptación)

Aspecto	Especificación
Nombre	Acceder al borrado de temporizadores, a través de URL, sin privilegios suficientes
Tipo de prueba	Aceptación
Criticidad	Alta
Precondiciones	Existe un usuario con rol readOnly y se ha ingresado al sistema con su nombre de usuario.
Escenario	El usuario accede al borrado de un temporizador existente a través de la URL.
Postcondiciones	Se muestra un mensaje de error y se redirige a la pantalla con la información del temporizador.

Cuadro C.35: Borrado de temporizadores sin privilegios suficientes

C.1.15. Activación de controladores

Aspecto	Especificación
Nombre	El controlador da de alta las válvulas (sin temporizadores)
Tipo de prueba	Aceptación
Criticidad	Media
Precondiciones	El controlador de válvulas tiene una o varias válvulas conectadas No tiene configurado ningún temporizador
Escenario	Se conecta el controlador.
Postcondiciones	Se crean (ó actualizan) las válvulas en el servicio web. El listado de temporizadores de cada una de esas válvulas debe aparece vacio. No se deben fijar alarmas en el controlador de válvulas.

Cuadro C.36: Activación del controlador

Aspecto	**Especificación**
Nombre	El controlador da de alta las válvulas (con temporizadores)
Tipo de prueba	Aceptación
Criticidad	Alta
Precondiciones	El controlador de válvulas tiene una o varias válvulas conectadas Tiene configurado varios temporizadores (almacenado en la base de datos de la aplicación Java)
Escenario	Se conecta el controlador.
Postcondiciones	Se crean (ó actualizan) las válvulas en el servicio web de la aplicación de control. Los temporizadores deben aparecer en las válvulas correspondientes desde el servicio web Respecto a las alarmas, se debe fijar primero la mas cercana en el tiempo al momento actual y, después de activarse esa, las siguientes. Las alarmas deben provocar las correspondientes aperturas y cierres de las válvulas.

Cuadro C.37: Activación del controlador con temporizadores

Bibliografía

[Ada15] Kevin MacG. Adams. *Non-functional Requirements in Systems Analysis and Design*. eng. 1st. Cham: Springer International Publishing, 2015.

[Apa20a] Apache. *Create a data-model*. 2020. URL: https://freemarker.apache.org/docs/pgui_quickstart_createdatamodel.html.

[Apa20b] Apache. *Embedded Derby*. 2020. URL: http://db.apache.org/derby/papers/DerbyTut/embedded_intro.html.

[Apa20c] Apache. *What is Apache Derby*. 2020. URL: https://db.apache.org/derby/#What+is+Apache+Derby%3F.

[Apa20d] Apache. *What is Apache FreeMarker?* 2020. URL: https://freemarker.apache.org/.

[Ard18] Arduino. *What is Arduino*. 2018. URL: https://www.arduino.cc/en/Guide/Introduction.

[Ard20a] Arduino. *Arduino MKR WiFi 1010*. 2020. URL: https://store.arduino.cc/arduino-mkr-wifi-1010.

[Ard20b] Arduino. *Introduction to the Arduino Board*. 2020. URL: https://www.arduino.cc/en/reference/board.

[Ard20c] Arduino. *WiFi.lowPowerMode() reference*. 2020. URL: https://www.arduino.cc/en/Reference/WiFiNINALowPowerMode.

[aWO20a] aWOT. *aWOT*. 2020. URL: https://awot.net/.

[aWO20b] aWOT. *aWOT Reference*. 2020. URL: https://awot.net/en/3x/api.html.

[BA09] Danny Múnera Blanca Alicia Correa Juan Fernando Eusse y José Edinson Aedo. «UML2SC: Herramienta para el diseño de sistemas electrónicos complejos utilizando los lenguajes UML y SystemC». En: (2009).

[Bar03] Douglas K. Barry. «Chapter 3 - Service-Oriented Architectures and Web Services». En: *Web Services and Service-Oriented Architectures*. Ed. por Douglas K. Barry. Morgan Kaufmann, 2003, págs. 17-35.

[Ber94] T. Berners-Lee. *Request for Comments: 1630: Universal Resource Identifiers in WWW*. 1994. URL: https://tools.ietf.org/html/rfc1630.

[BFF96] T. Berners-Lee, R. Fielding y H. Frystyk. *Request for Comments 1945: Hypertext Transfer Protocol – HTTP/1.0*. 1996. URL: https://tools.ietf.org/html/rfc1945#section-3.2.

[Bir20] Rain Bird. *Válvulas*. 2020. URL: https://www.rainbird.es/profesionales/productos/valvulas.

[Bra14] Tim Bray. *Request for Comments 7159: The JavaScript Object Notation (JSON) Data Interchange Format*. 2014. URL: https://tools.ietf.org/html/rfc7159.

[Bur92] Steve Burbeck. *Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller (MVC)*. 1992. URL: https://web.archive.org/web/20120429161935/http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html.

[Bus01] Frank Buschmann. *Pattern-oriented software architecture : a system of patterns*. Wiley series in software design patterns. John Wiley y Sons, 2001.

[Cal+08] Ben Caldwell y col. *Web Content Accessibility Guidelines (WCAG) 2.0*. 2008. URL: https://www.w3.org/TR/WCAG20/.

[Cod90] Edgar F. Codd. *The relational model for database management : version 2*. 1rst ed. reprint. with corrections. Addison-Wesley, 1990. ISBN: 0-201-14192-2.

[Con20] The World Wide Web Consortium. *HTML & CSS*. 2020. URL: https://www.w3.org/standards/webdesign/htmlcss#whatcss.

[cPU18] Klaus Olsen (chair), Meile Posthuma y Stephanie Ulrich. *Certified Tester: Foundation Level Syllabus*. International Software Testing Qualifications Board, 2018. URL: https://www.istqb.org/downloads/send/2-foundation-level-documents/281-istqb-ctfl-syllabus-2018-v3-1.html.

[De17] Brajesh. De. *API Management An Architect's Guide to Developing and Managing APIs for Your Organization*. First edition. Apress, 2017.

[Def88] United States Department of Defense. *Military Standard: Defense System Software Development - DOD-STD-2167A*. 1988. URL: http://everyspec.com/DoD/DoD-STD/download.php?spec=DOD-STD-2167A.008470.pdf.

[Des20] Desconocido. *Arduino: Driver L298N y Bluetooth HC-06*. 2020. URL: https://store.arduino.cc/arduino-mkr-wifi-1010.

[DL01] D.D. Defense y Space Science Library. *Systems Engineering Fundamentals*. 2001. ISBN: 9781537703466. URL: https://ocw.mit.edu/courses/aeronautics-and-astronautics/16-885j-aircraft-systems-engineering-fall-2005/readings/sefguide_01_01.pdf.

[Dog18] Fernando Doglio. *REST API development with Node.js*. 2.ª ed. Apress, 2018.

[Fie00] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. University of California, 2000.

[HB04] Hugo Haas y Allen Brown. *Web Services Glossary*. 2004. URL: https://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/#webservice.

[Hib20] Hibernate. *Hibernate ORM*. 2020. URL: http://hibernate.org/orm/.

[JF98] Ralph E. Johnson y Brian Foote. *Designing Reusable Classes: Journal of Object-Oriented Programming*. Department of Computer Science. University of Illinois. 1998. URL: https://www.researchgate.net/profile/Brian_Foote2/publication/215446177_Designing_Reusable_Classes/links/54d4eb8f0cf2464758069812.pdf.

[Kar+15] Isak Karabegovic y col. *The application of service robots for logistics in manufacturing processes*. 2015. URL: http://apem-journal.org/Archives/2015/APEM10-4_185-194.pdf.

[KH16] Isak Karabegovic y Ermin Husak. *CHINA AS A LEADING COUNTRY IN THE WORLD IN AUTOMATION OF AUTOMOTIVE INDUSTRY MANUFACTURING PROCESSES*. 2016. URL: http://www.mvm.fink.rs/Journal/Archive/2016/2016V42N3/Karabegovic_et_all/Karabegovic_et_all_Paper.pdf.

[Kos+11] Alexander Kossiakoff y col. *Systems engineering : principles and practice*. 2nd. John Wiley y Sons, 2011.

[Kra16] JÃ¶rg Krause. *Introducing Bootstrap 4*. Apress, 2016.

[LB13] Thierry Louvel Jéromeand Templier y Thierry Boileau. *Restlet in action : developing RESTful web APIs in Java*. Shelter Island: Manning, 2013. ISBN: 9781935182344.

[Lio04] Guillermo Castañón Lion. *Automatización del riego*. 2004. URL: https://www.mapa.gob.es/ministerio/pags/biblioteca/revistas/pdf_Agri/Agri_2004_860_202_205.pdf.

[Mah+20] Mohd Saiful Azimi Mahmud y col. *Robotics and Automation in Agriculture: Present and Future Applications*. 2020. URL: http://arqiipubl.com/ojs/index.php/AMS_Journal/article/download/130/88.

[Ora20] Oracle. *Java JDBC API*. 2020. URL: https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/.

[PP06] Petraq Papajorgji y Panos Pardalos. *Software Engineering Techniques Applied to Agricultural Systems*. 2.ª ed. Springer, 2006.

[Rai20] Rain. *Electroválvulas*. 2020. URL: https://www.rain.it/en/professionisti/elettrovalvole.

[RGB97] Ed. R. Fielding, J. Gettys y T. Berners-Lee. *Request for Comments: 2068: Hypertext Transfer Protocol – HTTP/1.1*. 1997. URL: https://tools.ietf.org/html/rfc2068.

[Tal19] *Restlet Framework | Overview*. Inglés. Talend. 2019. URL: https://restlet.talend.com/ (visitado el 31 de enero de 2020).

[Tal20a] Talend. *Connectors*. 2020. URL: https://restlet.talend.com/documentation/user-guide/2.4/core/base/connectors.

[Tal20b] Talend. *Resource Overview: Introduction*. 2020. URL: https://restlet.talend.com/documentation/user-guide/2.4/core/resource/overview.

[Tal20c] Talend. *Restlet Framework*. 2020. URL: https://restlet.talend.com/.

[Tal20d] Talend. *Routing Overview: Introduction*. 2020. URL: https://restlet.talend.com/documentation/user-guide/2.4/core/routing/overview.

[TB12] Cesare Pautasso Thomas Erl Benjamin Carlyle y Raj Balasubramanian. *SOA with REST: Principles, Patterns & Constraints for Building Enterprise Solutions with REST*. Prentice Hall, 2012. Cap. 5.

[tea20a] Bootstrap team. *Accessibility*. 2020. URL: https://getbootstrap.com/docs/4.5/getting-started/accessibility/.

[tea20b] Bootstrap team. *Browsers and devices*. 2020. URL: https://getbootstrap.com/docs/4.5/getting-started/browsers-devices/.

[Teo+11] Toby Teorey y col. *1 - Introduction*. Morgan Kaufmann, 2011, págs. 1-11.

[uBl19] u-Blox. *NINA-W10 series: Stand-alone Multiradio modules - Data sheet*. 2019. URL: https://content.arduino.cc/assets/Arduino_NINA-W10_DataSheet_%28UBX-17065507%29.pdf.

[WAM11] John-David Warren, Josh Adams y Harald Molle. *Arduino Robotics*. Apress, 2011.