UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

# Desarrollo de aplicación web para gestionar módulos inteligentes para la construcción de jardines

Trabajo Fin de Grado

**Grado en Ingeniería Informática**

**Autor**: Gil Rodríguez, Carlos

**Tutor**: Poza Luján, Jose Luís

**Cotutor**: Posadas Yagüe, Juan Luís

2015-2016

A todas esas personas que
siempre han confiado en mí,
en mis posibilidades, y que
sobre todo, me han apoyado.

# Resumen

En la actualidad, y cada vez más frecuentemente, se está expandiendo la necesidad de disponer de aplicaciones o **servicios web** enfocados a un ámbito específico. En este caso, el de los **jardines inteligentes**. Se va a llevar a cabo todo el proceso de creación de un servicio web **Java** mediante el framework **Google Web Toolkit** (**GWT**) cuyo objetivo será el poder llevar el control de este tipo de sistemas y gestionar la **información** que le rodea, como la proporcionada por **sensores** acoplados a **módulos inteligentes**.

**Palabras clave:** jardín inteligente, módulos inteligentes, sensores, información, servicio web, Java, Google Web Toolkit.

# Índice de contenidos

# Índice de ilustraciones

# Índice de tablas

# 1 Introducción

En este documento se va a poder leer todo el proceso de creación de un servicio web orientado a la gestión de los módulos inteligentes que componen cualquier jardín inteligente. Desde una investigación para comparar los productos relacionados ya existentes hasta las pruebas con el producto final, pasando por una minuciosa especificación de necesidades que tendrá que abarcar el servicio, el diseño de cada una de las partes que lo compondrán y el posterior desarrollo del mismo.

Actualmente la mayor parte del mercado de desarrollo de aplicaciones en el ámbito de controlar sistemas inteligentes propios, está enfocada en los dispositivos móviles ya que a priori son los dispositivos más adecuados para llevar la gestión de estos sistemas, pero no hay por qué olvidar el hacer aplicaciones para ordenador, ya que estos te permiten muchas más posibilidades a la hora de desarrollarlas.

Una vez realizado el estudio de las aplicaciones ya existentes, se procederá a hacer una comparativa de las mismas, para así poder obtener información sobre que tipo de características son las más comunes en este ámbito.

En ese punto el lector ya se podrá hacer una idea de que podrá realizar con el servicio web, así como conocer con que tecnologías se va a desarrollar.

Con los requisitos ya puesto sobre la mesa, se le dará paso al diseño del sistema, con diferentes diagramas explicativos de cada uno de los puntos, explicaciones de como se va a emplear el software elegido y en general, la estructura de la que dispondrá el servicio web.

Finalmente se explicará paso a paso la implementación del sistema, dando así a conocer gran parte del código Java desarrollado y que función tendrá cada parte de la estructura creada en el análisis anterior.

Para que el lector vaya entrando en materia, el primer capítulo va a ser una puesta en escena del entorno en el que se empleará el servicio web, así como de la relación que cada vez va siendo más fuerte entre las tecnologías y la vida cuotidiana de las personas.

# 2 Entorno

## 2.1 Entorno de realización

La jardinería, en todas sus amplias ramas, ha estado presente en la vida de las personas desde las edades más antiguas. Bien como necesidad, para abastecerse de alimentos, o como actividad de entretenimiento.

Con el paso de los años, en paralelo con la aparición de nuevas tecnologías, se ha extendido una forma de jardinería autónoma e inteligente. Casos como el riego automático de empresas relacionadas con el cultivo o de jardines de casas particulares son claros ejemplos.

El principal beneficio de esta adaptación de la jardinería con las nuevas tecnologías es bastante claro: el ahorro tanto de tiempo como de recursos. Y es que como en la mayoría de ámbitos, las personas utilizan todo lo necesario para ahorrarse tiempo y recursos en tareas costosas.

Más específicamente, en el ámbito de la jardinería y la agricultura, el regadío automático, que es lo que todo el mundo conoce por haberlo visto en grandes superficies agrícolas o en los jardines de nuestras propias ciudades, evita que tenga que ir una o varias personas a regar dichas superficies.

Las tecnologías avanzan y avanzan, y con ellas, los métodos para realizar acciones cotidianas como regar una planta, controlar su humedad, iluminación, temperatura... Desde hace un tiempo se escucha mucho el término Internet Of Things (IoT) [1], nombre que se le atribuye a la creación de un sistema autónomo y automático compuesto por objetos con sensores/actuadores presentes en el hogar (o en sus exteriores), los cuales podrán ser controlados todos a la vez desde un dispositivo electrónico (Smartphone, Tablet, Pc...).

A raíz del término IoT, aparece otro término, Web Of Things (WoT), el cual se puede definir como una capa interior del IoT, y que es la encargada de proporcionar al usuario una interfaz visual con la que poder gestionar toda la información proporcionada por el sistema inteligente.

En el caso de nuestro proyecto, vamos a implementar un servicio web desde el que se podrá gestionar toda la información de un sistema de jardín inteligente, así como la de todos los sensores y actuadores que este contenga.

## 2.2 Sistemas similares

Una vez conocemos el entorno en el que se va a desarrollar nuestro proyecto, vamos a realizar un estudio del mercado de las aplicaciones web para navegador enfocadas a gestionar la información de un jardín inteligente.

Debido a la coincidencia temporal de expansión de este tipo de sistemas y de las aplicaciones móviles (Android,iOS...), la mayoría del mercado existente se basa en la venta de *paquetes* compuestos por los dispositivos necesarios más una aplicación móvil para gestionarlos. En algunos de estos casos, también se ofrece un servicio web para los navegadores, que son los que vamos a analizar.

El objetivo de este estudio es analizar las diferentes opciones y obtener las características que ha de tener nuestro servicio web.

### 2.2.1 NetAQUA

En este servicio podemos observar a simple vista que la interfaz gráfica no ha sido el punto fuerte, ya que debido a la utilización del mismo color para todos los títulos y la falta de claridad a la hora de la separación de los diferentes componentes, conllevan a que el usuario se sienta abrumado por la falta de orden.

Por otro lado, dispone de la suficiente cantidad de información para que el usuario tenga a simple vista una visión global del estado del sistema.

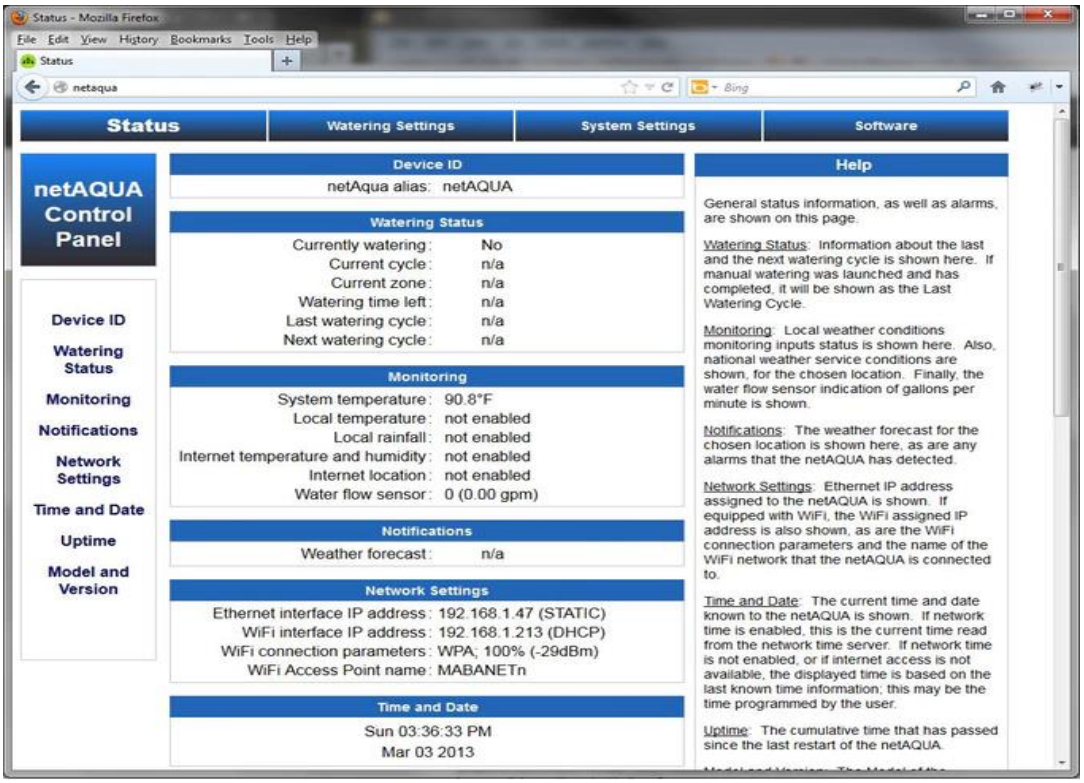



**Ilustración 1 - Interfaz del servicio web NetAQUA [1]**

NetAQUA aún no es un sistema que exista como tal, ya que se trata de un proyecto *crowfunding*. Tiene como única función controlar el regadío automático, mediante un dispositivo central que es el que lo gestiona todo. Además de la interfaz web, tiene un panel LCD que, en caso de no disponer de Internet, se pueden realizar las acciones más básicas.

### 2.2.2 BlueSpray

Esta interfaz se basa en una visión aérea del lugar donde esté implantado el sistema inteligente, facilitando así la elección del nodo del que queremos obtener la información.

En el panel lateral hay diferentes botones para la activación/desactivación de funciones como el riego automático a una hora fija, detectar la lluvia, detectar el flujo de agua en el sistema o poner el sistema en *stand-by*.

En el panel superior se encuentran los diferentes menús de configuración, calendario e histórico de acciones.

[1] https://www.kickstarter.com/projects/netaqua/netaqua-web-enabled-irrigation-controller/description

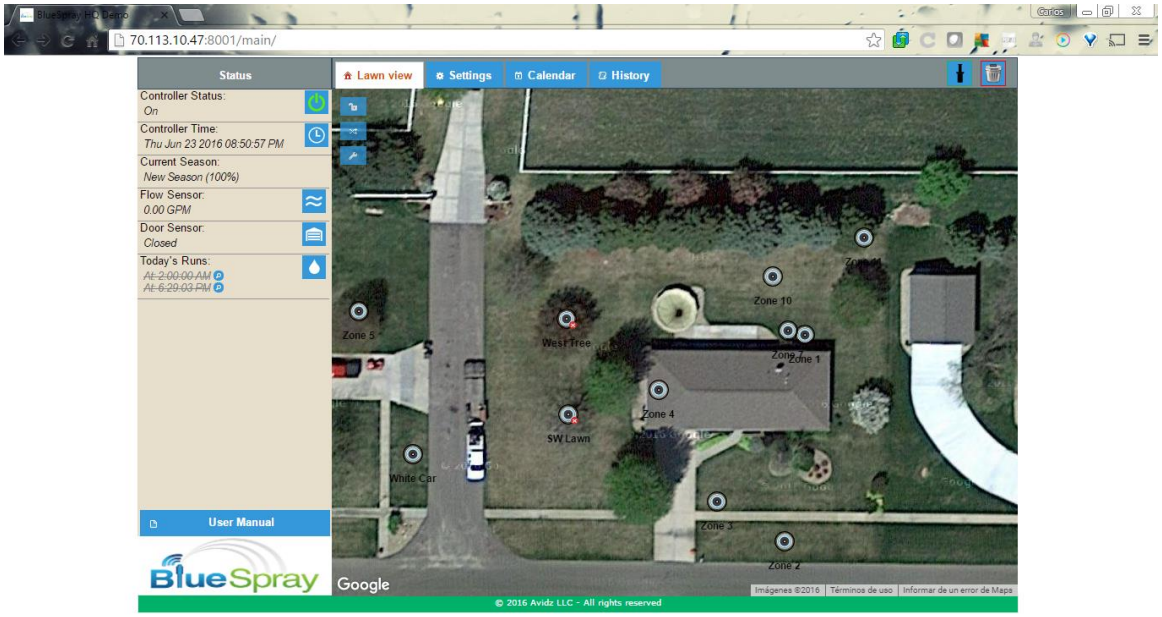



**Ilustración 2 - Interfaz del servicio web BlueSpray [2]**

En este caso, al tratarse ya de un producto inmerso en este mercado, BlueSpray es un producto más profesional, con la posibilidad de llevar el control sobre diversos sensores/actuadores. [2]

En este caso, al tratarse ya de un producto inmerso en este mercado, BlueSpray es un producto más profesional, con la posibilidad de llevar el control sobre diversos sensores/actuadores.

Como es frecuente en estos casos, se compone de un dispositivo central que canaliza las conexiones entre sensores/actuadores y la interfaz gráfica del usuario, y además dispone de su propia API REST para poder crearte tus propios servicios web.

### 2.2.3 Koubachi

El apartado gráfico en este servicio web es muy notable, partiendo de la simplicidad general de un escritorio personal vacío, puedes ir añadiendo tantas plantas como quieras, con o sin sensores. Una vez añadidas las plantas deseadas, cada una de ellas dispone de varios menús muy intuitivos con los que puedes observar la información de los sensores y actuadores o leer información específica del tipo de planta que se ha elegido.

Es notable el gran desarrollo que hay detrás de esta interfaz, ya que tiene colores muy adecuados, toda la información ordenada y una cantidad adecuada de botones con lo que interactuar.

Este sistema inteligente se basa en la interacción de la interfaz, bien vía web o mediante alguna de sus aplicaciones móviles, con los sensores acoplados a cada una de las plantas que queremos monitorizar.

Está enfocado al cultivo de plantas, ya que dispone de una gran base de datos con muchos tipos de planta y mucha información de cada una de ellas.

[2] https://www.bluespray.net/bin/demo

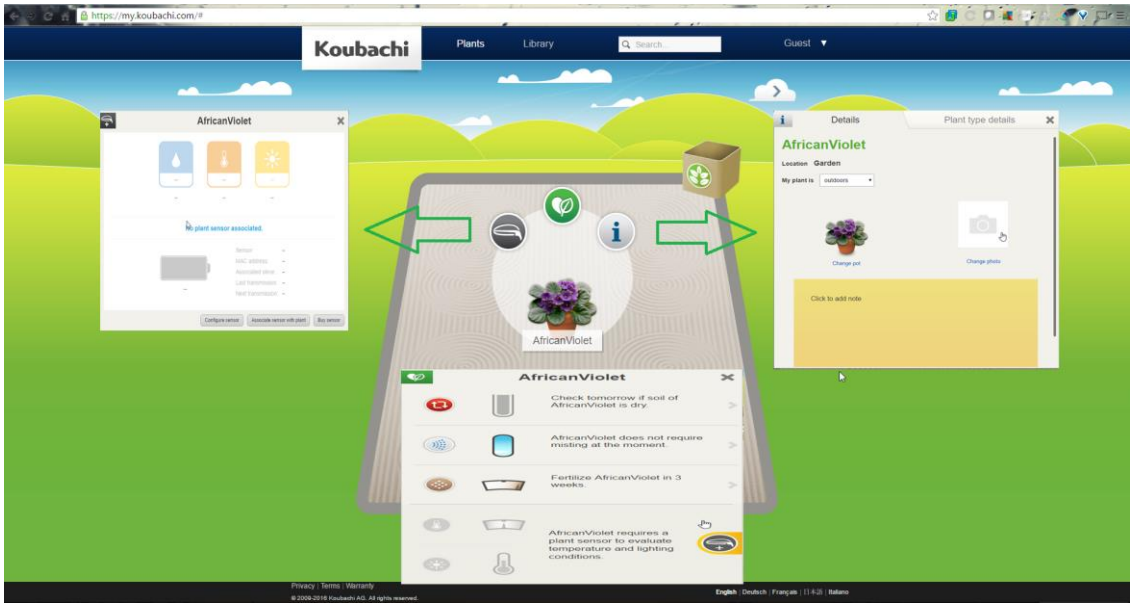



Ilustración 3 - Interfaz del servicio web Koubachi[3]

Gracias a que hay que indicarle el tipo de planta, y junto a los valores obtenidos por los sensores, el sistema te aconseja de cuando debes realizar las diferentes acciones de cuidado (regar, fertilizar…), además de disponer de un sistema de alertas para cuando alguno de los valores de sus sensores (luz, humedad, temperatura) no tiene un valor dentro de su umbral más óptimo.

## 2.3 Análisis

Tras conocer varias soluciones existentes al problema que queremos solucionar, se va a proceder a realizar un análisis comparativo de las mismas.

Con éste análisis finalizado, se podrá obtener tanto características obligatorias que ha de cumplir el proyecto como características secundarias u opcionales.

### 2.3.1 Análisis cuantitativo

Primero se analizan las características técnicas de los diferentes sistemas, es decir, todas aquellas características objetivas y medibles.

#### *2.3.1.1 Características generales*

- **Sistema**: Nombre del sistema
- **Tipo producto**: Si se trata de un servicio web ofrecido por un paquete o es un servicio web independiente.
- **Navegadores compatibles**: Navegadores en los que funciona correctamente el servicio web.
- **Software extra**: Necesidad de instalar software adicional para el correcto funcionamiento.
- **Internet**: Necesidad de tener conexión a Internet para el funcionamiento.
- **WI-FI**: Posibilidad de conectar dispositivos mediante WI-FI.
- **Servidor propio**: Posibilidad de implantar el servicio web en tu propio servidor web.
- **Precio**: Precio de venta al público del sistema.

[3] https://my.koubachi.com/

| Sistema | Tipo producto | Navegadores compatibles | Software extra | Internet | WI-FI | Servidor propio | Precio |
|---|---|---|---|---|---|---|---|
| NetAQUA | Paquete (proyecto crowfunding) | Todos los comunes | No | Sí | Sí | No | 200 € (1) |
| BlueSpray | Paquete comercial | Todos los comunes | No | Sí | Sí | No | 200 € |
| Koubachi | Paquete comercial | Todos los comunes | No | Sí | Sí | No | (2) |

Tabla 1 - Análisis cuantitativo (general)

(1)Posibilidad elegir cuánto pagas (crowfunding)

(2)Precio no disponible en web

#### *2.3.1.2 Características funcionales*

- **Sistema**: Nombre del sistema
- **Sensores caseros**: Posibilidad de añadir al servicio web tus propios dispositivos con sensores (Manualmente/Automáticamente).
- **Vista disponible**: Vista de datos general disponible / Vista de datos por nodo disponible.
- **Actuadores**: Posibilidad de dar órdenes a actuadores.
- **Localización**: Ver en un mapa la posición de nuestros nodos.
- **Historial**: Disponible un historial de registro de la información obtenida por los diferentes sensores.
- **Login**: Sistema de login usuario/contraseña para la seguridad.

| Sistema | Sensores caseros | | Vista disponible | | Actuadores | Localización | Historial | Login |
|---|---|---|---|---|---|---|---|---|
| | *Auto.* | *Manual* | *General* | *Nodo* | | | | |
| NetAQUA | | | | | | | | |
| BlueSpray | | | | | | | | |
| Koubachi | | | | | | | | |

Tabla 2 - Análisis cuantitativo (funcional)

Sí que cumple funcionalidad

No cumple funcionalidad

### 2.3.2 Análisis cualitativo

Por otro lado también analizaremos la parte subjetiva de los servicios web, aquella que engloba las características que dependen de la opinión personal.

- **Primera impresión**: Calidad de la primera impresión del servicio web.
- **Curva de aprendizaje**: Relación tiempo/dificultad para manejar bien el sistema.
- **Intuición**: Nivel de intuición de los botones existentes y de las tareas que has de realizar para llevar a cabo una funcionalidad.
- **Aspecto visual:** Calidad de la interfaz gráfica.
- **Orden**: Orden y cohesión de la información mostrada.

Para cada una de las características descritas, se le va a asignar un valor único a cada uno de los sistemas, siendo 3 la mejor puntuación, 2 la puntuación intermedia y 1 la peor puntuación.

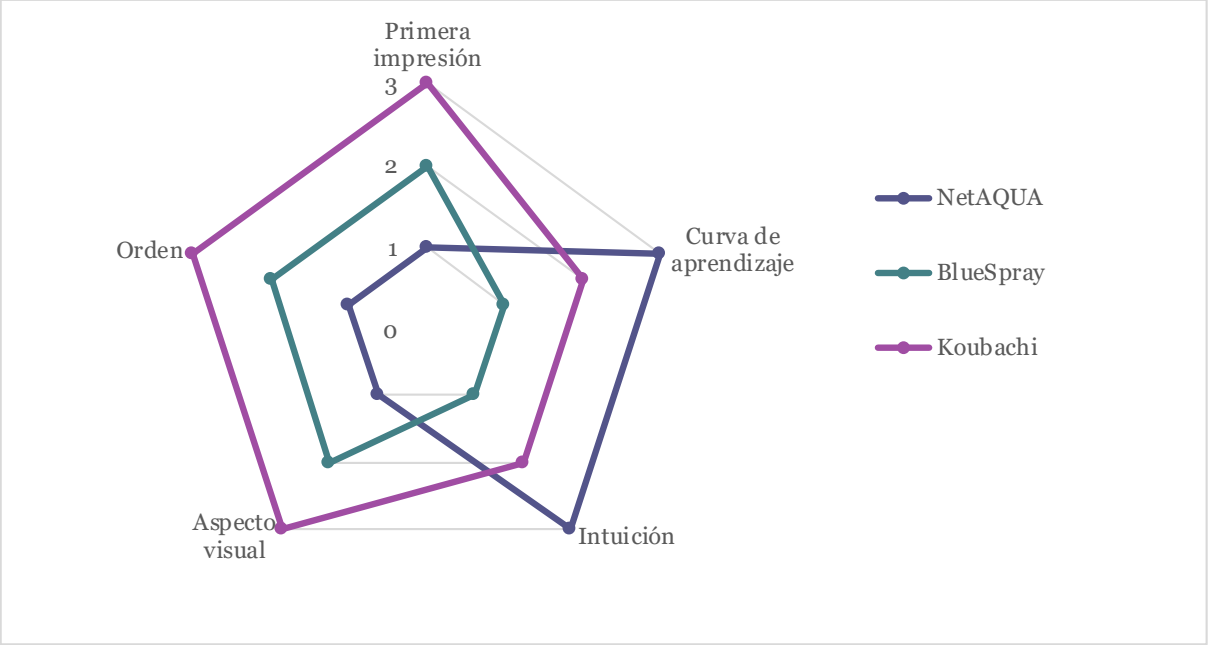



Tabla 3 - Análisis cualitativo

## 2.4 Síntesis

Ahora que ya se ha realizado las comparativas pertinentes, se van a poder extraer las características necesarias para el proyecto.

Gracias al análisis cuantitativo se obtienen los requisitos mínimos que ha de tener el servicio web, así como otros requisitos opcionales. Por otro lado, el análisis cualitativo ayuda a la hora de saber qué mejorar para hacer el servicio más competitivo respecto a los ya existentes.

A continuación, se listan las características mínimas que ha de tener el proyecto, codificándolas mediante CaXX, donde XX representa el número de la característica.

Ca01: Control de acceso al servicio web mediante user/pass.

Ca02: Conexiones: Posibilidad de agregar una dirección IP de la misma red WI-FI como nodo padre de los sensores.

Ca03: Posibilidad de lectura a tiempo real y por historial de los valores de los sensores de cada uno de los nodos agregados al servicio web.

Ca04: Posibilidad de mandar órdenes a actuadores de los nodos agregados al servicio web.

## 2.5 Tecnología a emplear

Con todas éstas características obtenidas y analizadas, ya existe una idea global de en qué va a consistir el proyecto.

El servicio web que se implementará tendrá que ser el encargado de la gestión de la información proporcionada por diferentes sensores y actuadores presentes en cualquier tipo de dispositivo preparado para ello (Arduino [2], Raspberry [3]...), es decir, cualquier

dispositivo que tenga la posibilidad de enviar su información mediante el protocolo HTTP [4], y más específicamente, mediante la arquitectura de comunicación REST [5].

Se ha decidido, por los motivos que veremos más adelante, utilizar Java [6] como lenguaje de programación del servicio web. Además, como se quiere orientar a la web, se utilizará el framework GWT [7], que facilita la utilización de la tecnología AJAX [8].

Por otra parte, para la persistencia de los datos, habrá un sistema de gestión de bases de datos relacionales MySQL [9].

Todo ello se realizará con uno de los IDE más conocidos en el entorno de la programación, Eclipse [10].

### 2.5.1 Java

Se ha elegido usar Java principalmente por las siguientes características:

- **Universalidad e independencia de la plataforma**: Ya que se pretende que exista una total independencia del navegador y el sistema operativo en el que se lance el servicio web, Java es el candidato ideal debido a su gran transportabilidad. Y es que basta con compilar una vez el proyecto implementado en este lenguaje, para luego poder hacerlo correr en cualquier plataforma que tenga un intérprete de Java (PC, Smartphone, Tablet...).
  Además, debido a su gran expansión en el ámbito de la programación (está presente en 7 mil millones de dispositivos, y va a más) , existe una gran cantidad de bibliotecas estándar para realizar operaciones complejas como las conexiones con bases de datos, o el envío de datos en red por protocolos estándar.

- **Seguridad**: Gracias a su orientación a objetos, y consecuente facilidad de aprendizaje, se producen menos errores de programación y consecuentemente, existe una mayor seguridad frente a ataques maliciosos o daños en el sistema.

- **Captura de excepciones**: Java dispone de un sistema de captura de errores cuando cualquier método que utilizamos produzca un error de cualquier tipo. En vez de que el servicio quede inutilizado cuando se produzca un error en la ejecución del código, se lanzará el código correspondiente al punto de captura de la excepción que se le haya implementado. Esto facilita que no se tenga que preocupar de diseñar un sistema de códigos de error.

### 2.5.2 GWT

Google Web Toolkit (GWT) es un framework que es especialmente útil para la creación del futuro servicio web, ya que tiene como objetivo simplificar la complejidad que tiene crear aplicaciones AJAX.

Puede ser empleado desde cualquier IDE de programación, y su tarea es convertir todo el código Java del servicio web a HTML [11] + JavaScript [12].

Puesto que tanto HTML como JavaScript están aceptados por la gran mayoría de navegadores web actuales, con el uso de ésta tecnología nos aseguramos la independencia del navegador que buscamos.

Por otra parte, también cabe destacar que dispone de un sistema de comunicación cliente-servidor RPC que una vez aprendido su funcionamiento, es muy útil en el tema del mantenimiento de las clases, ya que asegura una total independencia de la parte cliente y la

parte servidor. Si se quisiera cambiar la forma de implementar algún método del servidor, sólo se tendría que modificar el mismo, sin verse afectada la parte cliente, que es totalmente ajena a la lógica de negocio.

Finalmente, a diferencia de otros frameworks o formas de programar, GWT tiene un modo denominado *Hosted Mode* que permite realizar los cambios que queramos de la parte cliente del proyecto, guardar, y directamente observar los cambios visualmente, todo ello sin necesidad de compilar el proyecto cada vez. Esto es una herramienta realmente útil ya que una vez todo bien configurado, se gana mucho tiempo.

## 2.6 Conclusiones

A lo largo de éste capítulo, primero se ha hecho un estudio de mercado de las diferentes opciones existentes para nuestro problema, observando que dicho mercado es muy escaso, debido a la gran popularidad actual de controlarlo todo con dispositivos móviles.

Además, de los escasos servicios web centrados en gestionar la información de un jardín inteligente, gran parte de ellos trabajan única y exclusivamente con el sistema que acompaña al servicio web en el paquete de venta, por lo que nos deja un rango más amplio de competitividad para el servicio web.

También se han obtenido las características más deseables que ha de tener el servicio, así como los requisitos y funcionalidades que ha de cumplir: Tendrá que tener la posibilidad de agregar cualquier tipo de dispositivo que transmita la información de sus sensores a través de una red WI-FI, deberá poderse leer e interpretar dicha información, y por último, se tendrá que poder realizar actuaciones sobre los actuadores de los mismos dispositivos, en caso de contar con ellos.

Por último, se ha decidido utilizar el lenguaje Java mediante el framework GWT.

# 3 Especificación de requisitos

## 3.1 Introducción

En ésta sección del proyecto vamos a proceder a formalizar los requisitos obtenidos en la sección anterior, creando así el documento de Especificación de Requisitos Software (ERS). Para ello nos basaremos en la estructura del estándar IEEE-830.

### 3.1.1 Propósito

Con esta especificación de requisitos, se pretende determinar fielmente las necesidades que va a satisfacer el servicio web.

Las personas que decidan montar un jardín inteligente, y usar el servicio web como gestor de la información, tienen que saber qué van a poder realizar con el producto y qué no.

Ésta sección es la encargada de mantener una comunicación entre todas las personas que se vean afectadas por el desarrollo del servicio web, ya sean los clientes que lo vayan a utilizar, o los desarrolladores, para marcar unas pautas de necesidad.

### 3.1.2 Ámbito del sistema

InGarduino es un servicio web genérico enfocado al control de un sistema de jardín inteligente compuesto por nodos independientes. Cada uno de estos nodos dispondrá de sensores (de humedad, de temperatura, de iluminación...) y actuadores.

### 3.1.3 Definiciones, acrónimos y abreviaturas

***Definiciones***

- **Nodo***:* Punto de conexión entre dos o más elementos de un circuito.
- **Sensor***:* Objeto capaz de detectar magnitudes físicas y transformarlas en variables eléctricas.
- **Actuador***:* Dispositivo capaz de transformar energía eléctrica en la activación de un proceso con la finalidad de generar un efecto sobre un proceso automatizado

***Acrónimos***

- **IP**: Internet Protocol.
- **URL**: Uniform Resource Locator.
- **REST**: Representational State Transfer.

### 3.1.4 Personal Involucrado

| | |
|---|---|
| Nombre | Carlos Gil Rodríguez |
| Rol | Desarrollador servicio web |
| Categoría profesional | Estudiante |
| Responsabilidades | Diseño, desarrollo e implementación |
| Información de contacto | Cargiro1@inf.upv.es |
| Aprobación | - |

**Tabla 4 - Miembro Carlos Gil Rodríguez**

Definiciones: https://es.wikipedia.org

| | |
|---|---|
| Nombre | José Luis Poza Luján |
| Rol | Supervisor. |
| Categoría profesional | Profesor Contratado Doctor |
| Responsabilidades | Supervisar proyecto. |
| Información de contacto | joplu@disca.upv.es |
| Aprobación | - |

Tabla 5 - Miembro José Luís Poza Luján

### 3.1.5 Visión general del documento

Este documento se compone de tres partes: la introducción, la descripción general y los requisitos específicos.

En la primera se explica en qué va a consistir el proyecto, así como para qué sector está dirigido.

En la segunda, se verá a grandes rasgos y sin excesivo detalle cual va a ser la función de el servicio web InGarduino, que características ha de tener y qué funciones ha de cumplir.

Por último, en los requisitos específicos, se puntualizará cada una de las funciones que ha de cumplir el servicio para satisfacer los requisitos detallados anteriormente.

## 3.2 Descripción general

### 3.2.1 Perspectiva del producto

El servicio web InGarduino ha de facilitar la tarea de conocer el estado de los nodos agregados al mismo, ayudando así a la automatización del control del jardín inteligente.

Se trata de un sistema autónomo sin relación con ningún otro producto, pero sí cabe destacar que tiene que existir a priori un sistema de jardín inteligente ya implantado, al que el servicio web se le acopla.

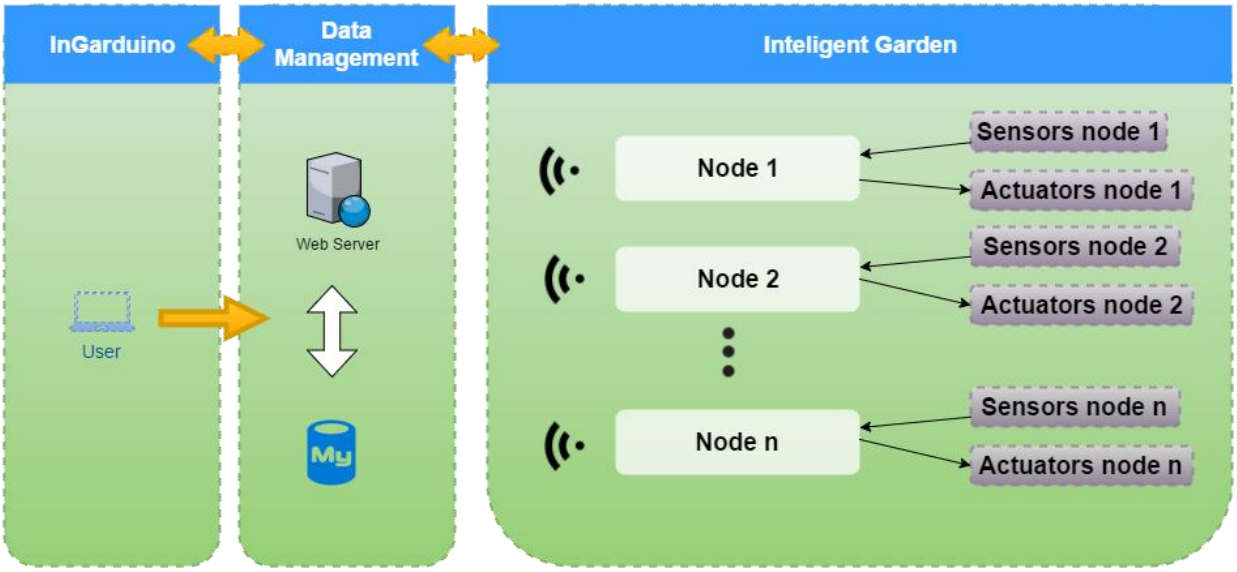



Ilustración 4 - Diagrama de bloques del proyecto[1]

En la imagen superior se puede observar la estructura del sistema en el que funcionará el servicio web. El usuario interactuará desde un ordenador personal con el servicio, el cual está alojado en un servidor web.

Este servidor web donde estará implantado el servicio, será el encargado de:

1. Recibir una petición por parte de la interfaz gráfica del servicio InGarduino.
2. Procesar la petición.
3. Leer datos de la base de datos en caso de que se requiera.
4. Leer datos de los nodos en caso de que se requiera.
5. Lanzar órdenes a los actuadores en caso de que se requiera.
6. Devolver los datos pedidos a la interfaz gráfica.

### 3.2.2 Funciones del producto

En este sub-apartado procedemos a citar las funcionalidades que el servicio ha de integrar:

**Gestión de módulos inteligentes para la construcción de jardines:** será capaz, tras el pertinente login del usuario en el servicio web, de agregar tantos nodos como desee, facilitando la dirección IP del nodo, la URL encargada de obtener la información de sus sensores mediante la técnica de comunicación REST, y la URL encargada de realizar las acciones sobre los actuadores. Así se consigue una total independencia del tipo de nodo (Arduino, Raspberry, PC...).

Una vez agregados los nodos, el usuario podrá gestionar la monitorización de los mismos, pudiendo así observar los diferentes niveles de los sensores en el jardín (humedad relativa, temperatura, iluminación...) y consecuentemente llevar a cabo una serie de acciones en el mismo nodo del que se ha obtenido la información.

### 3.2.3 Características de los usuarios

La interfaz gráfica del servicio web está orientada a ser lo más simple e intuitica posible, por lo que a la hora de navegar por él, el usuario no requiere de tener conocimientos avanzados ni de informática ni de electrónica.

A la hora de agregar nuevos nodos, sí que es cierto que hacen falta saber datos que un usuario de a pie no tiene porque saber. Esto se solucionaría pudiendo acceder a la implementación del funcionamiento de los nodos, para poder forzar que tengan alguna clase Java que implemente una interfaz que nos facilite estos datos. Pero debido a la autonomía entre sistema inteligente e InGarduino, esto no es posible a no ser que el jardín sea un sistema OpenSource.

La solución a este problema sería contactar con la persona encargada de la implantación del jardín inteligente para que nos facilite los datos necesarios.

### 3.2.4 Restricciones generales

El ordenador desde el que se acceda a InGarduino ha de pertenecer a la misma red doméstica a la que pertenezca el servidor web. Esto podría solucionarse alojando el proyecto GWT en algún sistema de hosting en Internet.

[1]Fuente: Elaboración propia

Ya que hemos elegido la opción de utilizar un servidor en la misma red, el único requisito que ha de tener el servidor, es que tiene que ser capaz de poder desplegar aplicaciones desarrolladas en Java.

Las limitaciones de qué tipo de magnitudes podremos observar desde el servicio las marcarán los tipos de sensores que dispongan los nodos.

### 3.2.5 Suposiciones y dependencias

El proyecto se a desarrollado con la tecnología GWT, con el objetivo de abarcar el máximo número de navegadores y sistemas operativos posible, ya que el lenguaje JavaScript, que es lo que finalmente se ejecuta en el navegador, está admitido por una inmensa mayoría de éstos.

A lo largo del desarrollo se ha empleado Windows 10 y Google Chrome para realizar las diferentes pruebas. También se ha probado el servicio web base en otros navegadores como Internet Explorer, Edge (cabe destacar que el funcionamiento de estos dos navegadores es diferente ya que tienen diferente motor) y Mozilla Firefox.

La base de datos relacional que hemos empleado en el proyecto es de tipo MySQL, y lo más aconsejable es emplear este tipo debido a su facilidad de conexión y gestión, pero cambiar este tipo a MSSQL u Oracle no debería ser un camino angosto, ya que las sentencias SQL utilizadas se han hecho genéricas.

## 3.3 Requisitos específicos

### 3.3.1 Requisitos de interfaces externas

#### *3.3.1.1 Interfaces del usuario*

- ***Usabilidad y sencillez***

| | |
|---|---|
| Código de requisito | RE-01 |
| Nombre | Usabilidad y sencillez |
| Inputs | - |
| Outputs | - |
| Proceso | La interfaz gráfica ha de cumplir todos los requisitos especificados, así como cumplir un mínimo grado de usabilidad, sencillez y claridad. |
| Restricciones | - |

Tabla 6 - Requisito de usabilidad y sencillez

#### *3.3.1.2 Interfaces hardware*

- ***Dispositivo de interacción***

| | |
|---|---|
| Código de requisito | RE-02 |
| Nombre | Dispositivo de interacción |
| Inputs | - |
| Outputs | - |
| Proceso | Para poder acceder al servicio web, hace falta un ordenador personal con un sistema operativo instalado, un navegador y conexión a Internet. |
| Restricciones | - |

Tabla 7 - Requisito de dispositivo de interacció

### 3.3.2 Requisitos funcionales

En éste sub-apartado se va a detallar minuciosamente cada uno de los requisitos funcionales que ha de cumplir el sistema.

Todos y cada uno de ellos se han especificado teniendo en cuenta que cada uno de ellos ha de pasar un test, es decir, que cada requisito se debe poder demostrar si está implantado o no en el sistema.

- ***Registro en el sistema***

| | |
|---|---|
| Código de requisito | RE-03 |
| Nombre | Registro en el sistema |
| Inputs | Nombre de usuario y password |
| Outputs | Cierra formulario de registro y loguea al usuario. |
| Proceso | Si los datos son válidos, el usuario/password es creado en base de datos, para así que pueda acceder cuando quiera. Acto seguido se loguea al usuario automáticamente. |
| Restricciones | Para que el sistema reconozca que se trata de un registro nuevo, al username le añadiremos el prefijo "CU-". |

Tabla 8 - Requisito de registro de usuario

- ***Login en el sistema***

| | |
|---|---|
| Código de requisito | RE-04 |
| Nombre | Login en el sistema |
| Inputs | Nombre de usuario y password |
| Outputs | Cierra formulario de registro y loguea al usuario |
| Proceso | El sistema debe verificar el username/pass en base de datos. Si la comprobación es correcta se cierra el formulario y se loguea el usuario automáticamente. Si algún parámetro no corresponde, aparece mensaje de error y se mantiene el formulario de login. |
| Restricciones | - |

Tabla 9 - Requisito de login del sistema

- ***Logout del sistema***

| | |
|---|---|
| Código de requisito | RE-05 |
| Nombre | Logout del sistema |
| Inputs | - |
| Outputs | Se cierra la sesión y aparece el formulario de login. |
| Proceso | El sistema debe disponer de algún método para que el usuario pueda cerrar sesión. |
| Restricciones | -El usuario tiene que haberse logueado con anterioridad. |

Tabla 10 - Requisito de logout del sistema

- ***Vista general nodos***

| | |
|---|---|
| Código de requisito | RE-06 |
| Nombre | Vista general de nodos |
| Inputs | - |
| Outputs | Vista con todos los nodos agregados |
| Proceso | El sistema debe poder mostrar una vista en la que se visualicen todos los nodos agregados. |

| | |
|---|---|
| Restricciones | El usuario tiene que estar logueado en el sistema. |

Tabla 11 - Requisito de vista general de nodos

- ***Agregar nuevo nodo***

| | |
|---|---|
| Código de requisito | RE-07 |
| Nombre | Agregar nuevo nodo |
| Inputs | Datos necesarios del nuevo nodo |
| Outputs | Aparece el nuevo nodo en la vista de los nodos. |
| Proceso | El sistema debe registrar el nuevo nodo en base de datos y asociarlo al usuario que lo ha añadido, para que así cada usuario tenga su vista de nodos personalizada. |
| Restricciones | El usuario tiene que estar logueado en el sistema |

Tabla 12 - Requisito de agregar nuevo nodo

- ***Leer información de sensores***

| | |
|---|---|
| Código de requisito | RE-08 |
| Nombre | Leer información de sensores |
| Inputs | Pinchar con el ratón sobre el nodo del que se quiere obtener la información. |
| Outputs | Aparece un listado con los diferentes sensores del nodo y sus valores correspondientes |
| Proceso | El sistema debe poder mostrar, para cada nodo, los valores actualizados de sus sensores. |
| Restricciones | El usuario tiene que estar logueado en el sistema |

Tabla 13 - Requisito de lectura de los valores de los sensores

- ***Realizar acciones sobre nodos***

| | |
|---|---|
| Código de requisito | RE-09 |
| Nombre | Realizar acciones sobre nodos |
| Inputs | Pinchar con el ratón sobre el nodo al que se le quiere enviar la acción, luego elegir la misma. |
| Outputs | Se realiza la acción sobre el nodo. |
| Proceso | El sistema debe poder realizar las acciones pertinentes sobre los actuadores de cada uno de los nodos. |
| Restricciones | El usuario tiene que estar logueado en el sistema |

Tabla 14 - Requisito de acción sobre los actuadores de los nodos

### 3.3.3 Requisitos no funcionales

A continuación se van a describir aquellos requisitos que no afectan a si el sistema cumple las funcionalidades expuestas o no.

#### *3.3.3.1 Requisitos de rendimiento*

En lo que concierne a los requisitos relacionados con el rendimiento tanto del ordenador personal desde el que ser va a acceder al servicio web como del servicio web en sí, cabe destacar que gracias a la tecnología GWT empleada, no se necesitan una gran cantidad de recursos para que el funcionamiento sea óptimo.

Como puede observarse en la siguiente imagen, el servicio web no necesita ni la mitad de memoria RAM que uno de los servicios ya existentes y analizados.

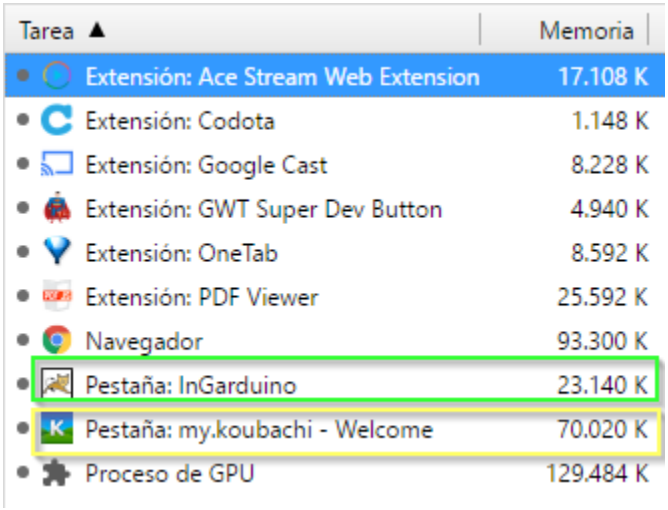



**Ilustración 5 - Comparación de memoria RAM empleada**

El sistema está desarrollado de forma que sólo puede existir un usuario logueado a la vez, puesto que el control de diversos usuarios, sus correspondientes hilos concurrentes, control de sesiones, y demás consecuencias habrían demorado mucho el desarrollo, pero sería una buena mejora en un futuro.

#### *3.3.3.2 Seguridad*

Dado que es muy difícil limitar o restringir a las personas para que no tengan acceso al servicio web, deberán implementarse ciertas medidas de seguridad con el objetivo de proporcionar formas seguras de trasmisión de datos así como de proteger el contenido confidencial.

Para esto, la principal herramienta de seguridad es la necesidad de realizar un inicio de sesión en el servicio antes de poder realizar cualquier acción. Si no has hecho login previamente, intentes lo que intentes hacer en el servicio te redirigirá a la pantalla de login.

Si el login es correcto, al usuario se le dará permisos para poder acceder a todas las funcionalidades, en cambio si no, aparecerá un mensaje de error y se redirigirá al formulario de entrada.

Para que un login se considere válido, previamente el usuario se tiene que haber registrado en el sistema. Cuando un usuario se registra en InGarduino, se guarda en base de datos tanto su username como su pass cifrada mediante el algoritmo BCrypt.

Éste ultimo punto es muy importante, ya que si se guardaran las contraseñas como texto sin cifrar, y la base datos fuera objetivo de un ataque malicioso, todas las cuentas estarían expuestas al atacante.

Además, gracias al haber empleado un patrón de diseño Model-View-Presenter (MVP), en la URL de el servicio no se declaran ningún tipo de parámetros o atributos, sino que para cada una de las vistas implementadas, es la misma URL hagas la acción que hagas dentro de ella. Esto nos soluciona un problema muy común últimamente que es el de la inyección de datos en la URL para producir ataques maliciosos.

#### *3.3.3.3 Mantenibilidad*

Gracias también a la buena organización de clases que te permite usar el patrón MVP, la tarea de mantenibilidad no es demasiado compleja.

Si en algún momento salta alguna excepción desconocida, será muy fácil observar el error, ir a la clase donde se produce y solucionarlo.

Tareas como actualización de clases o métodos, se llevan a cabo rápidamente

#### *3.3.3.4 Portabilidad*

Desarrollar en el lenguaje Java, y que GWT te lo convierta todo a JavaScript tiene un principal beneficio: universalidad. Y es que gracias a ésta característica, InGarduino se puede ejecutar en cualquier tipo de sistema operativo y desde cualquier navegador (salvando que en algún navegador, la interfaz gráfica se descuadre por culpa de la no aceptación de ciertos estilos CSS).

Por lo que podemos considerar que el servicio se caracteriza por una gran portabilidad.

## 3.4 Conclusiones

El objetivo de este capítulo ha sido transformar las características generales que habíamos obtenido en el análisis anterior, en un documento estructurado por el estándar IEEE-830.

Primero se ha descrito el ámbito en el que se va a implantar el sistema del proyecto, luego se ha formalizado de manera general varios aspectos como características, requisitos y funcionalidades, para finalmente definir detalladamente y formalizar cada uno de ellos.

A continuación, ya que se conoce perfectamente todo lo que tiene que poder hacer el servicio web, se va a proceder a realizar el diseño del sistema.

Se ha formalizado cada una de las características, requisitos y funcionalidades que ha de tener el servicio web.

# 4 Diseño del sistema

## 4.1 Introducción

Una vez se tiene finalizado el documento de Especificación de Requisitos Software, es momento de empezar a plantear el diseño del servicio web que cumpla todos y cada uno de los requisitos de dicho documento.

Primero se va a llevar a cabo un análisis de todas las acciones que va a realizar el servicio mediante diagramas, para obtener una visión general del funcionamiento del sistema y de su base de datos.

Se dedicará un apartado a la toma de contacto con cada elemento software que vamos a tener que emplear a lo largo de todo el proyecto, con los cuales podremos desarrollar todos los casos de uso que tenga que cumplir el servicio. Después se describirá cada caso de uso con su correspondiente UML.

Finalmente mostraremos los primeros bocetos del futuro servicio web InGarduino.

## 4.2 Especificación general

La intención de este sub-apartado es mostrar de una forma general cual va a ser el funcionamiento del servicio web: la estructura que va a tener, los caminos funcionales de los que dispondrá un usuario dentro de él, la estructura de la base de datos relacional a emplear.

### 4.2.1 Estructura

Para la estructura de la implementación, vamos a emplear una arquitectura cliente-servidor con el objetivo de repartir los procesos a realizar.

La arquitectura cliente-servidor es un tipo de arquitectura productor-consumidor, donde el servidor actúa como productor y el cliente como consumidor.

La parte cliente, que lo normal es que sea el navegador desde el que se acceda al servicio web, es el encargado de llevar el control de la interfaz gráfica del servicio, además de ser el que realiza las llamadas al servidor dependiendo de que funcionalidad se desee y de los datos que disponga.

Por otra parte, la parte servidor está en constante espera de que le realicen llamadas, una o varias partes cliente a la vez, para luego cuando captura que le han realizado una petición, procesarla y devolver el resultado correspondiente. El servidor es el encargado de manejar la información de la base de datos, y puede tratase de cualquier dispositivo con capacidad de almacenamiento, conexión de red y capacidad de procesamiento de código y datos. Es decir, un servidor puede ser desde un micro-controlador tipo Raspberry Pi hasta un clúster gigante, pasando por dispositivos como un ordenador personal o un servidor web.

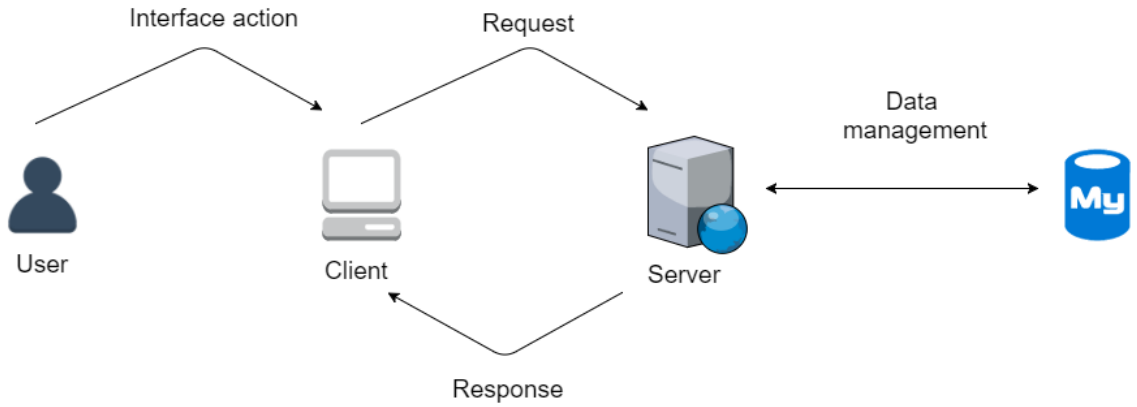



Ilustración 6 - Arquitectura cliente-servidor

Las ventajas que obtenemos al utilizar este tipo de arquitectura son las siguientes:

- **Mantenimiento del servicio más claro y sencillo**: En caso de disponer de varios ordenadores actuando como servidor, al tener las funciones y sus respectivas clases en lugares diferentes, se reducen mucho los problemas a la hora del mantenimiento (reparación, actualización...).
- **Seguridad**: Gracias a la creciente existencia de tecnologías en ese ámbito, las transacciones entre cliente-servidor son completamente seguras aplicando las medidas adecuadas.
- **Escalabilidad**: Como ya se ha mencionado, el servidor se puede componer de tantos ordenadores como el usuario desee, bien para ganar en rendimiento o en seguridad por el tema de caídas del servidor.

### 4.2.2 Diagrama base de datos relacional

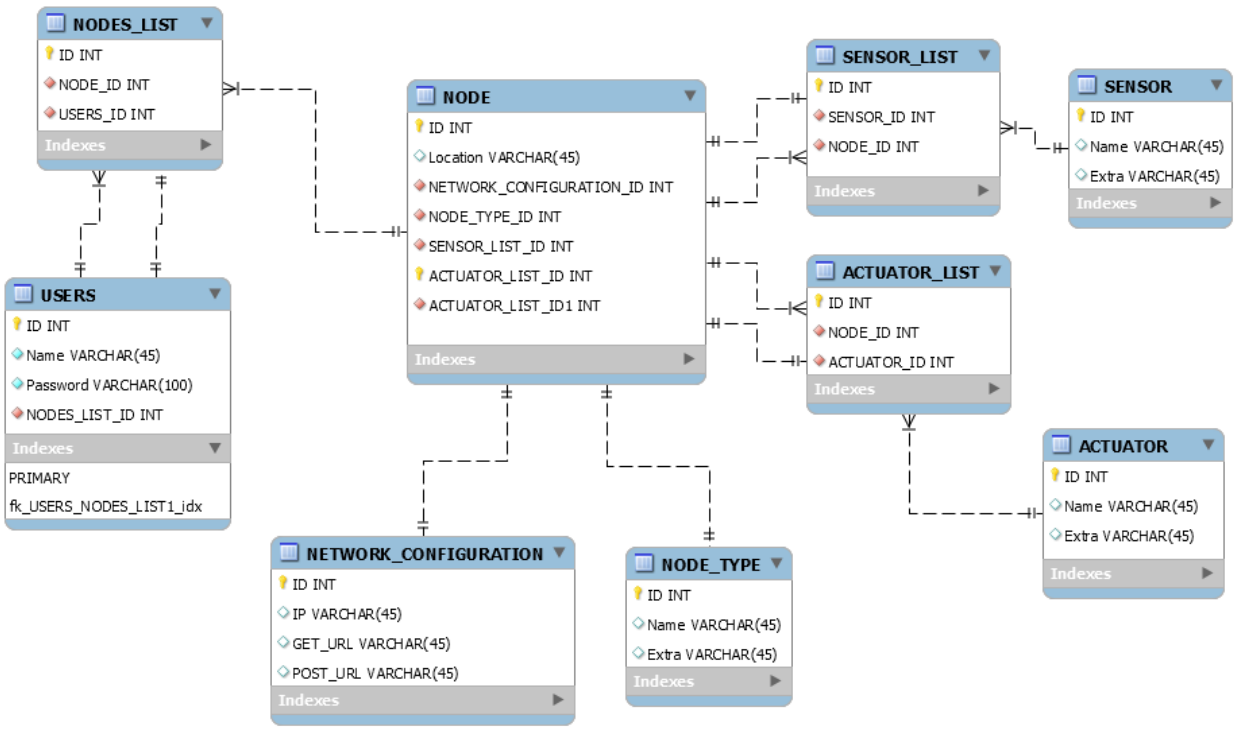



Ilustración 7 - Diagrama base de datos

Como se puede observar en el diagrama anterior, se parte de que cada usuario registrado en el sistema tendrá asociada una lista de nodos, que es la que le aparecerá en la vista principal.

Cada nodo a su vez, está compuesto por una configuración de red, un tipo de nodo, una lista de sensores y otra de actuadores.

### 4.2.3 Diagrama UML casos de uso

Con el siguiente diagrama UML se van a definir las diferentes acciones que va a poder realizar el usuario en el servicio web InGarduino.

Servirá de ayuda para ver de una forma simple las distintas formas de utilización del servicio.

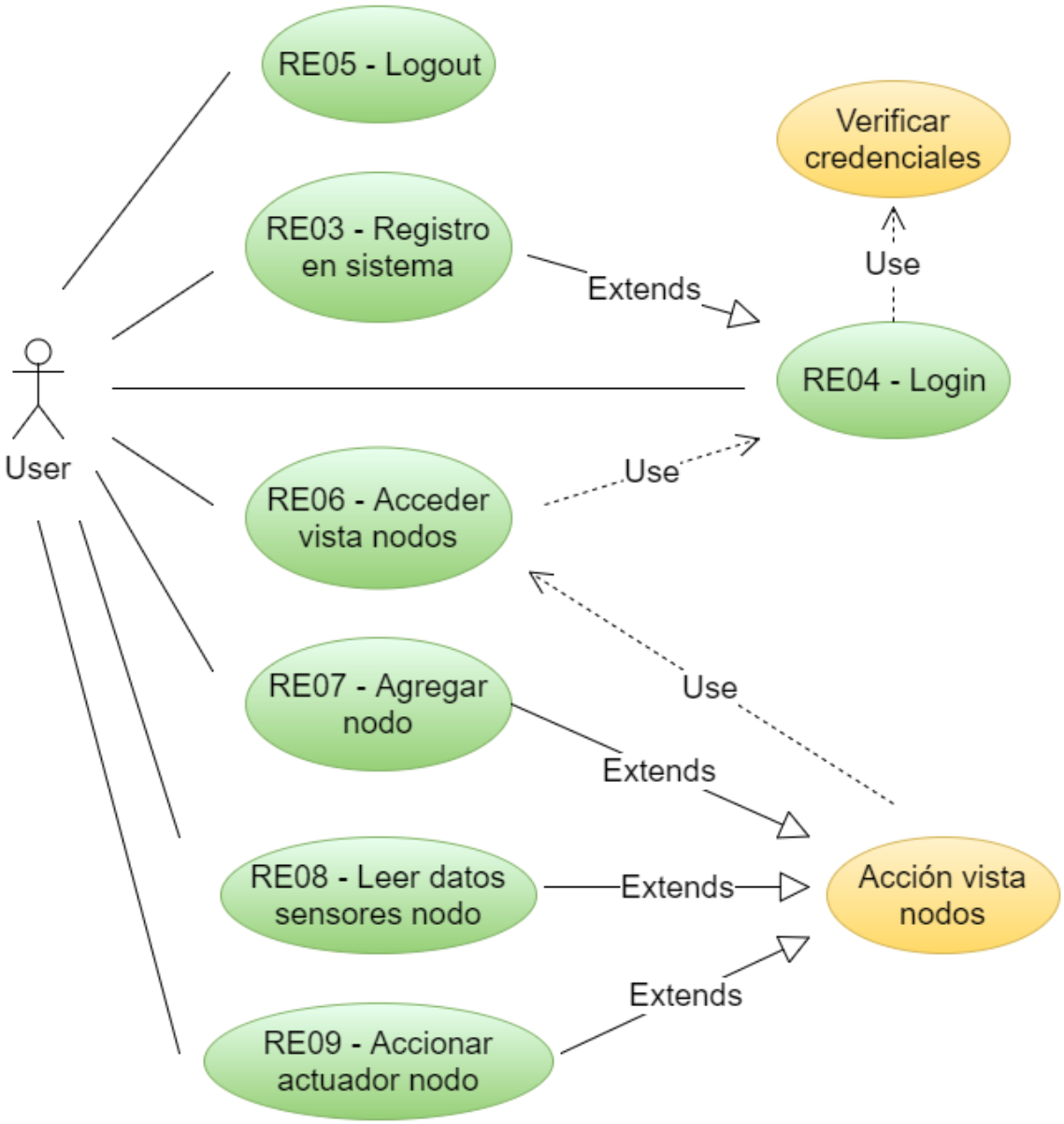



Ilustración 8 - Diagrama UML casos de uso

Como se puede observar, cada caso de uso corresponde a cada uno de los requisitos funcionales descritos en el apartado 3.3.2 Requisitos funcionales, haciendo referencia a cada uno de ellos mediante su código.

De esta manera nos aseguramos que cada requisito funcional, que son los que el servicio web tiene que cumplir, tiene asociada una acción dentro del servicio.

Consecuentemente, podemos afirmar que el servicio web va a cumplir con todos los requisitos funcionales exigidos.

## 4.3 Software empleado

### 4.3.1 Eclipse

El entorno de programación (IDE) que hemos elegido para desarrollar el sistema es Eclipse, ya que debido a su gran popularidad en el entorno de la programación mundial, dispone de una gran cantidad de documentación por parte de la comunidad y además es el entorno en el que mejor se integra GWT.

Para poder crear un proyecto GWT en Eclipse y posteriormente poder emplear todas sus funcionalidades, hace falta la instalación de su debido plugin, como se puede observar en las siguientes capturas de pantalla:

1. Descargamos e instalamos Eclipse IDE for Java EE Developers desde la página web oficial de Eclipse.

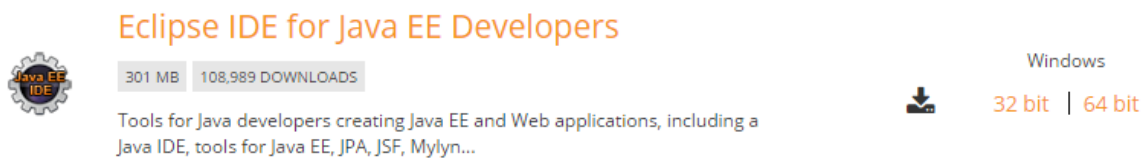



Ilustración 9 - Instalación Eclipse

2. Desde la página web oficial de GWT descargamos su SDK, y como se va a usar Eclipse, elegimos la opción de descargar también el plugin.


Ilustración 10 - Instalación GW

3. Crear nuevo proyecto desde File -> New -> Other -> Google -> Web Application Project.

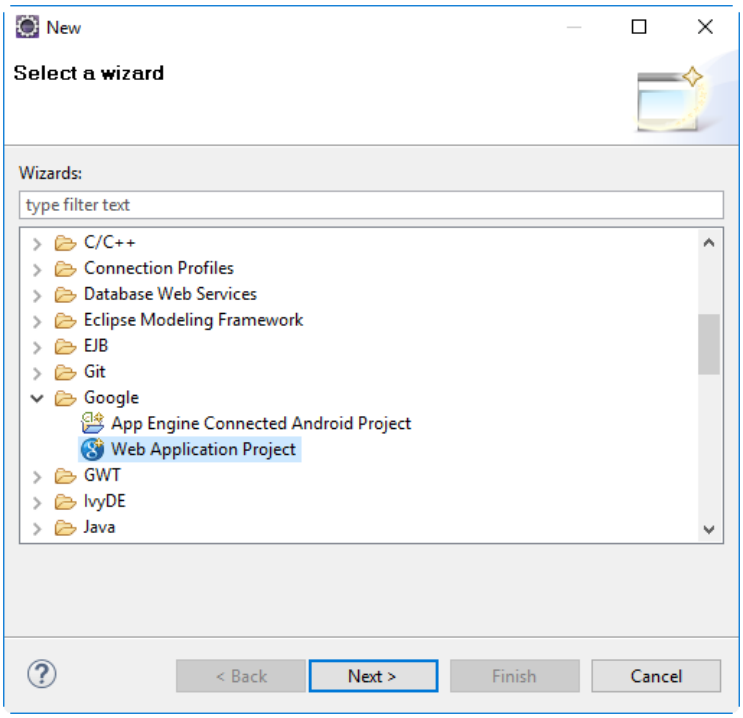



Ilustración 11 - Creación proyecto GWT

A continuación se muestra la jerarquía de paquetes creada por el proyecto GWT, que como hemos explicado anteriormente, está dividido por la parte cliente y por la parte servidor:

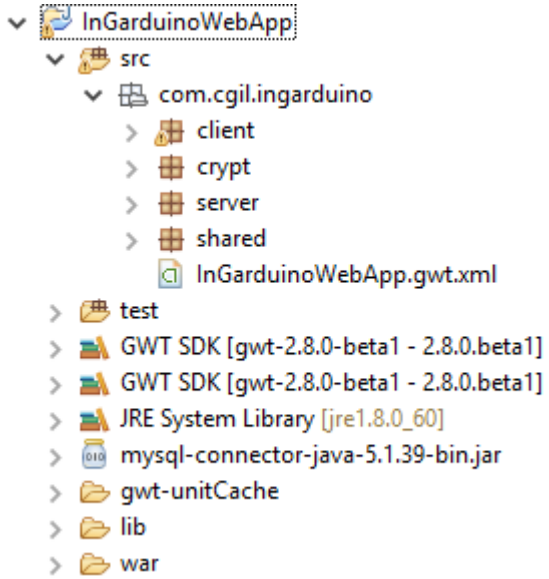



Ilustración 12 - Jerarquía paquetes proyecto GWT

### 4.3.2 Google Web Toolkit (GWT)

#### *4.3.2.1* AJAX

La técnica de programación AJAX (Asynchronous JavaScript And Xml) se trata de un conjunto de técnicas cuyo objetivo es la creación de páginas web dinámicas e interactivas.

Su principal característica es que debido a su conexión asíncrona constante con el servidor, a diferencia de las páginas web estáticas, no hace falta actualizarlas cuando se lleva a cabo alguna acción.

La forma en la que AJAX trabaja es la siguiente:

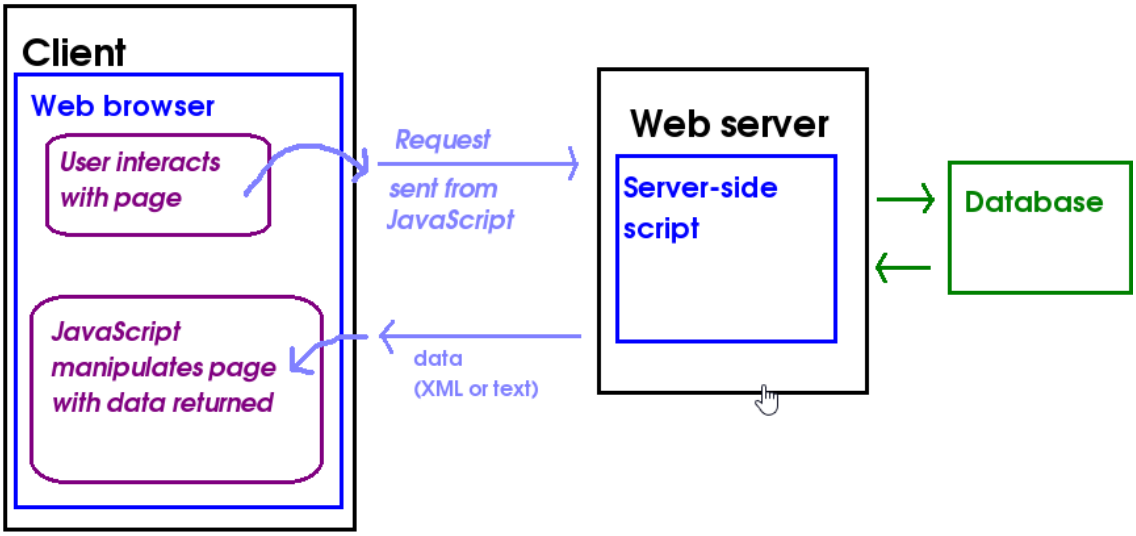



Ilustración 13 – AJAX[1]

Una vez aclarado el significado y funcionamiento de la tecnología AJAX, podemos adentrarnos en GWT. Se trata de un framework creado por Google en el año 2006 con el propósito de facilitar a los desarrolladores la creación de aplicaciones tipo AJAX.

A grandes rasgos, su *trabajo* consiste en transformar el código Java creado por el desarrollador en cualquier entorno de desarrollo, a HTML y JavaScript, con lo que logra que la aplicación creada pueda ser lanzada desde prácticamente cualquier navegador web.

Gracias a su extensa librería para la interfaz gráfica, no es necesario añadir librerías externas para poder crear un servicio web, pero sí que existen varias como SmartGWT, Sencha, Vaadin o GWT-Bootstrap.

Nosotros hemos decidido no utilizar librerías externas por dos motivos: ligereza del proyecto y que no se disponía del tiempo necesario para aprender a usar nuevas librerías.

Además, dispone de un modo de desarrollo llamado Hosted Mode para agilizar aún más los cambios en el código, evitándonos compilar cada vez. Esto lo veremos con más detalle en el futuro apartado de Pruebas.

#### 4.3.2.2 Patrón de diseño MVP

En la construcción de cualquier servicio o página web, la ordenación de las clases implementadoras cobra un papel muy importante a nivel de diseño, y es que si no se emplea algún método de separación de código por clases y por funciones del código, un proyecto básico puede convertirse en un proyecto muy difícil de mantener debido al desorden.

[1]Fuente: Google Images

Aquí es cuando entran en juego los patrones de diseño, que podrían definirse como el esqueleto de las soluciones a problemas comunes en el desarrollo de software. Es decir, una estructura a seguir para solucionar problemas de desarrollo software de contextos similares.

En el caso de creación de un proyecto GWT, desde Google aconsejan seguir el patrón MVP [13] (Model-View-Presenter) con el objetivo de conseguir el máximo desacoplamiento de clases posible.

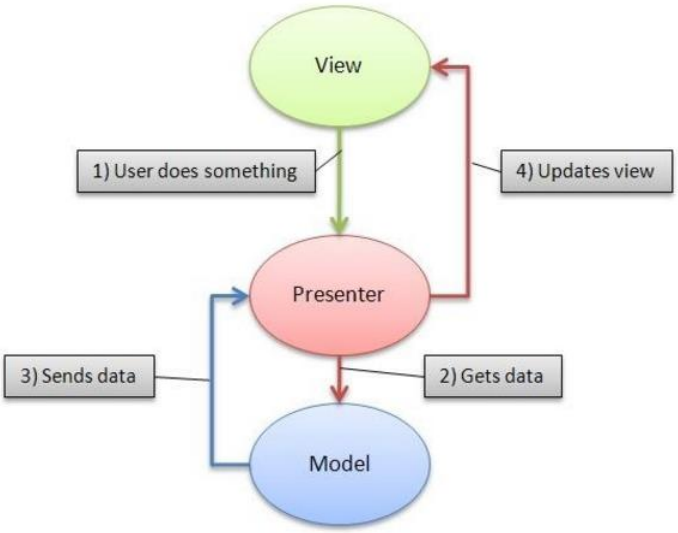



**Ilustración 14 - Patrón MVP[1]**

El funcionamiento de este patrón se basa en la fuerte interacción entre la vista (interfaz gráfica) y el presentador (manejador de la lógica), ya que se consigue que sean totalmente independientes, y que la vista no tenga porque saber nada de que se va a hacer con los datos introducidos por el usuario.

El presentador es el que se encarga de esta tarea. Recoge los datos y eventos que sucedan en la vista, y dependiendo de la lógica programada, hará unas acciones u otras con los datos del modelo para finalmente devolver a la vista los datos o acciones resultantes.

### 4.3.3 MySQL

Para controlar el apartado de persistencia de los datos del servicio web hemos elegido utilizar MySQL como base de datos relacional.

Además de la posibilidad de crear relaciones entre las diferentes tablas para conseguir una mayor eficiencia y seguridad, MySQL se caracteriza por ser un sistema de gestión multihilo y multiusuario, características que han hecho que sea uno de los motores de bases de datos más utilizados en el mundo del desarrollo software.

Otro punto a favor es su licencia, ya que esta base de datos se ofrece bajo licencia GNU GPL para los proyectos no corporativos y/o profesionales. Para este último caso habría que comprar la licencia correspondiente.

[1] Fuente: http://www.pearltrees.com/t/design-patterns/mvc-variants/id4558646

- ***Integración con Eclipse***

Para la correcta integración de la base de datos relacional MySQL con el entorno Eclipse, tenemos que descargarnos el conector correspondiente, en nuestro caso el MySQL Connector/J, el cual es una librería .jar que tenemos que establecer en la ruta raíz del proyecto.

A continuación hago referencia a un enlace web donde se explica paso a paso toda esta configuración y cómo agregar nuestra base de datos al entorno Eclipse.

https://www.zkoss.org/wiki/Setup_MySQL_DB_in_Eclipse

- ***MySQL Workbench***

Para la fácil gestión de la base de datos, hemos decidido instalarnos el software MySQLWorkbench, la cual es una herramienta muy recomendable debido a su sencillez de instalación y de configuración.

Con esta herramienta podremos desde crear nuevas bases de datos mediante scripts o diagramas, hasta monitorizar el rendimiento de la base de datos, pasando por realizar todo tipo de consultas, y añadir/modificar/eliminar datos con una interfaz gráfica muy intuitiva.

## 4.4 Mockups

En este último sub-apartado del diseño del sistema, vamos a proceder a mostrar diversos bocetos de las diferentes vistas que hay que tener en mente para el servicio web.

- ***Vista principal***

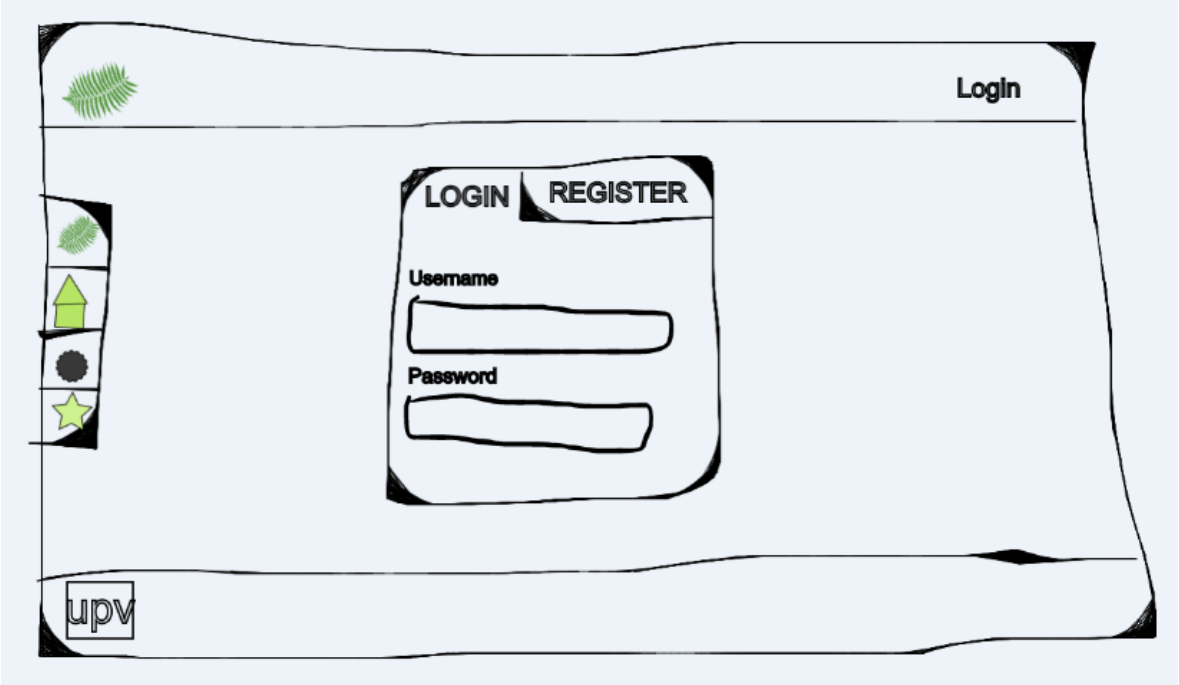



Ilustración 15 - Mockup vista principal

- ***Vista general de los nodos agregados***

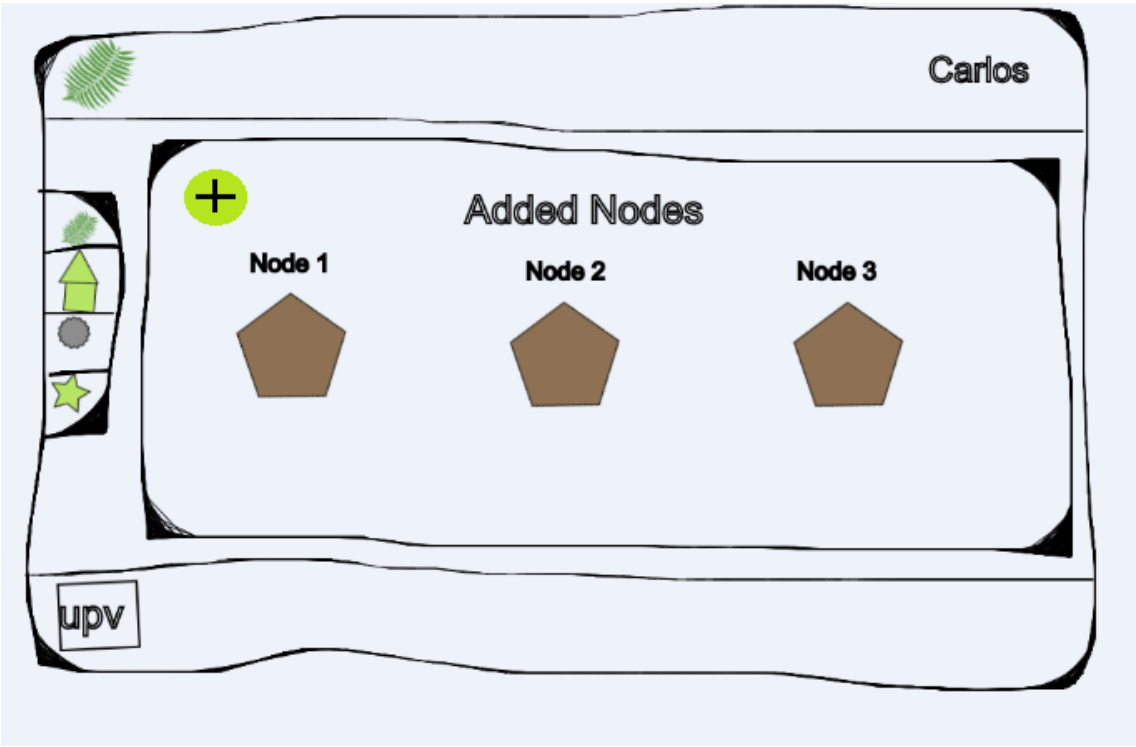



Ilustración 16 - Mockup vista general nodos

- ***Información de los sensores de un nodo***

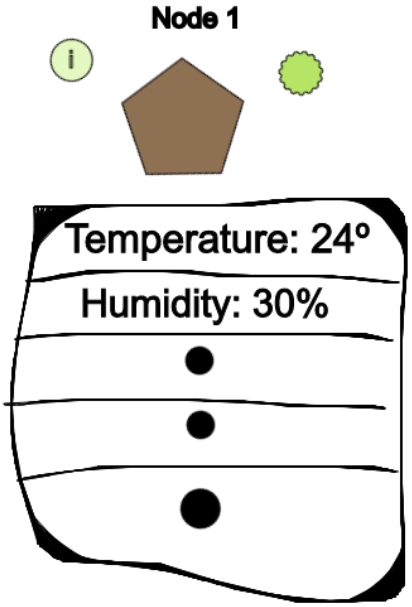



Ilustración 17 - Mockup sensores de un nodo

- ***Acciones sobre los actuadores de un nodo***

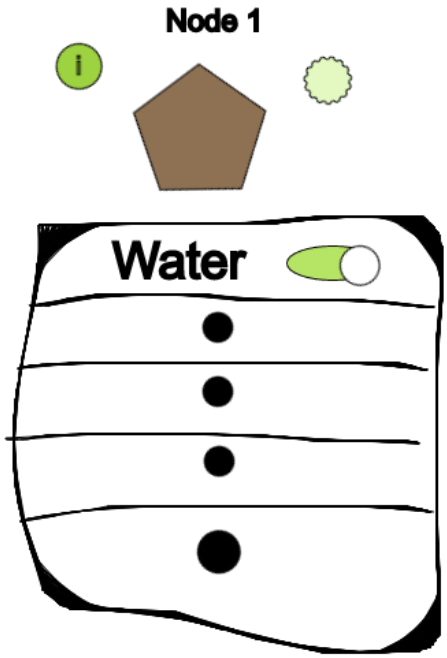



Ilustración 18 - Mockup actuadores de un nodo

## 4.5 Conclusiones

Una vez concluido este capitulo, ya tenemos unas ideas bien claras de cómo va a ser el servicio web, tanto visual como lógicamente.

Necesitaremos una interfaz gráfica similar a la vista en los bocetos anteriores, ya que cumple la sencillez que buscamos en nuestro servicio y además dispone de todas las funcionalidades exigidas por los requisitos obtenidos en el capítulo 3-Especificación de requisitos.

Dicha interfaz se implementará en la capa View del patrón de diseño Model-View-Presenter, el cual nos proporcionará una independencia total entre las clases encargadas de crear las interfaces visuales, las encargadas de la lógica de negocio, y las encargadas del modelo.

Esto nos permitirá obtener una estructura de clases muy ordenada para un mejor mantenimiento de las mismas, así como el desacoplamiento entre interfaz y lógica que buscamos para la universalidad del servicio.

Ahora que ya disponemos tanto de los requisitos del sistema como el diseño con el que vamos a llevar a cabo la implementación de los mismos, el siguiente capítulo se encargará de reunir todos los aspectos relacionados con la implementación, la implantación y la evaluación del producto.

# 5 Implementación, implantación y evaluación

## 5.1 Introducción

En este capítulo vamos a proceder a explicar el proceso de desarrollo del servicio web, mostrando la suficiente cantidad de código para que quede claro cómo están implementados los requisitos que hemos obtenido a lo largo de este documento.

Se empezará diferenciando entre el código generado para la parte cliente del servicio y la parte del servidor, para una vez aclarado este punto, explicar de una manera comprensible las partes más importantes del código.

Después de explicar las partes más significativas del desarrollo, se explicará el cómo implantar el servicio web en un servidor que nos permitirá acceder desde la misma red, y finalmente, se mostrarán diversas pruebas realizadas con InGarduino.

## 5.2 Implementación

### 5.2.1 Parte cliente

#### *5.2.1.1 Implementación patrón MVP*

En esta sección vamos a explicar la composición y estructura de clases de la parte cliente del servicio web, el cual hemos decidido estructurar con el patrón MVP, como bien habíamos descrito en el apartado 4.Diseño del sistema.

En términos generales, podemos resumir el comportamiento de un servicio GWT, en un sistema de vistas por las que se va accediendo a unas o a otras dependiendo de las acciones que realicemos en nuestra interfaz gráfica.

Cada una de las vistas tiene asociado un presentador, que es el encargado de manejar todos los datos e interacciones de la vista. Esta relación View-Presenter se consigue forzando a que cada una de las View sea una implementación de una interfaz generada por su Presenter, la cual dispondrá de todos los métodos necesarios para la correcta interacción usuario-interfaz gráfica.

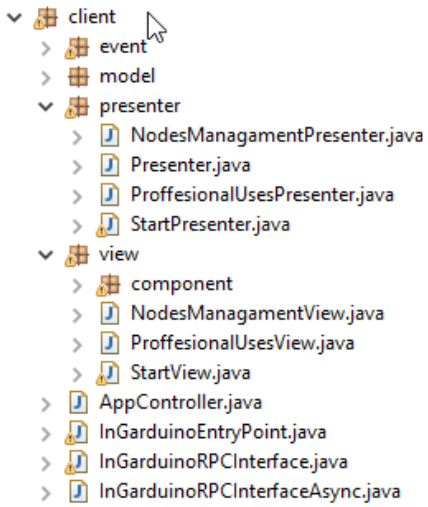



Ilustración 19 - Estructura de clases de la parte cliente

Como se puede observar en la ilustración 19, el package client está compuesto por diferentes sub-packages y clases:

- **Package event:** Clases cuyo objetivo es manejar las acciones a realizar cuando se producen determinadas acciones. Por ejemplo, manejar las acciones a realizar cuando se produce la acción de añadir un nuevo nodo al servicio.
- **Package model:** Clases encargadas de la gestión del acceso a la información de cada uno de nuestros objetos Java propios. Por ejemplo, la clase User.java encapsula toda la información que vayamos a necesitar sobre los usuarios.
- **Package presenter:** Clases del apartado Presenter del patrón MVP.
- **Package view**: Clases del apartado View del patrón MVP.
- **AppController.java**: Clase encargada del manejo de las diferentes vistas.
- **InGarduinoEntryPoint.java**: Punto de entrada del servicio web, donde se inicializa la primera vista a mostrar.
- **InGarduinoRPCInterface.java**: Interfaz de métodos disponibles en nuestras llamadas RPC. Este tema lo abordaremos más adelante.
- **InGarduinoRPCInterfaceAsync.java**: Versión asíncrona de la clase anterior.

Como ejemplo para explicar el funcionamiento de la interacción entre View y Presenter, vamos a utilizar las clases relacionadas con la vista inicial del servicio web, la que muestra la pantalla inicial donde dispondremos a realizar el login (StartView.java).

#### *5.2.1.2 Presenter*

Ya que cada una de las View va a ser una implementación de una interfaz generada por su correspondiente Presenter, vamos a comenzar explicando la estructura y el funcionamiento de este último.

Cada una de las clases Presenter será una implementación de la propia interfaz Presenter, la cual dispondrá de un método go(...) que será el encargado de llevar a cabo todas las acciones necesarias para ese Presenter en específico.

```
public interface Presenter {
      public abstract void go(final HasWidgets container);
}
```

Ilustración 20 - Interfaz Presenter

Dentro de la clase implementadora de esta interfaz, crearemos una nueva interfaz llamada Display que será la que tendrá que implementar la View asociada.

Esta interfaz Display contendrá todos los métodos necesarios para la interacción de la interfaz gráfica con la capa de persistencia de datos.

```
public class StartPresenter implements Presenter {

    public interface Display {
        HasMouseMoveHandlers getStartSymbolHoverable();
        HasClickHandlers getStartSymbolClickable();
        HasClickHandlers getPerfilSymbol();
        HasClickHandlers getLogoutSymbol();
        HasClickHandlers getUpvSymbol();
        HasClickHandlers getEtsinfSymbol();
        HasClickHandlers getContactSymbol();

        // Métodos para interactuar con el view
        //Login
        String getUserBoxValue();
        String getPassBoxValue();
        HasClickHandlers getLoginButton();
        void changeLoginVisibility(Boolean show);

        //Perfil
        void setLoggedLabelText(String username);

        Widget asWidget();

    }
```

Ilustración 21 - Interfaz Display

En esta ilustración observamos como creamos la sub-interfaz con todos los métodos que tendrá que implementar la View, desde obtener elementos clickables para luego realizar una acción u otra, hasta obtener/añadir la información de los diferentes campos de texto existentes.

Una vez está clara la estructura que ha de tener cada una de las clases implementadoras de Presenter, vamos a explica r su funcionamiento.

Además de tener la definición de la interfaz Display dentro de la clase StartPresenter.java, también nos tenemos que crear una instancia de la misma en una variable, que nos servirá para poder usar los métodos descritos dentro de Display como lo hacemos en la siguiente ilustración.

```
private final Display display;
public void bind() {
    //Accion al clik en login
    display.getLoginButton().addClickHandler(new ClickHandler() {

        public void onClick(ClickEvent event) {
    });
```

Ilustración 22 – Ejemplo uso instancia Display

Inicialmente habíamos dicho que nuestras clases Presenter debían implementar una interfaz que forzaba la implementación de un método go(...). En el caso del ejemplo StartPresenter.java, el método go(...) es el siguiente.

```
@Override
public void go(HasWidgets container) {
    checkLogin();
    bind();
    container.clear();
    container.add(display.asWidget());
}
```

Ilustración 23 - Implementación interfaz Presenter

Las acciones realizadas al llamar al método go(...) de la clase StartPresener.java son las siguientes:

1. Comprobar si existe un usuario con login en el sistema.
2. Declaración de usos de los métodos descritos en la interfaz Display.
3. Reiniciar el contenedor que nos pasan como parámetro,que será el Widget donde aparecerá representada nuestra StartView.java.
4. Añadir al dicho contenedor un nuevo Widget hijo, el Widget resultante de StartView.java.

#### *5.2.1.3 View*

Cada una de las View que hemos implementado en el servicio, la hemos creado mediante los Panels que ofrece el propio GWT en las librerías que ya incluye por defecto. Hemos decidido no utilizar librerías externas para la parte gráfica por dos motivos: el primero que no se disponía del tiempo suficiente para analizar todas las librerías que existen para ver cual se nos ajustaba más (hay muchas), y otra que al buscar un cierto grado de universalidad y facilidad de mantenimiento en el servicio, cuantas menos librerías externas al propio GWT metamos, mejor.

Además, estructurar las diferentes vistas empleando los propios Panels de GWT ofrece una claridad de las capas que componen la vista que no se obtiene con otros modos.

También disponemos de muchos objetos tipo *Widget* para agregar como hijos a los paneles, como Labels, Images, Buttons, ComboBoxes... por lo que simplemente con la instalación de GWT ya disponemos de todos los elementos necesarios para crear nuestra interfaz gráfica.

Todos los paneles, widgets, componentes, layouts de GWT son sencillos de implementar gracias a la gran cantidad de documentación que existe tanto por parte del propio Google como de la extensa comunidad de desarrolladores GWT.

El único punto en contra que se le puede atribuir a emplear esta forma de generar nuestra interfaz gráfica es el apartado visual, ya que los elementos de GWT tienen un aspecto muy básico, demasiado sencillo

Como solución a este último problema, hemos decidido emplear las hojas de estilo CSS (Cascading Style Sheets) para cada uno de los elementos a los que queramos cambiarles alguna parte o comportamiento visual.

A continuación, vamos a explicar el uso de los Panels GWT para el caso de la vista principal, StartView.java.

- ***Panels utilizados***

**SplitLayoutPanel**

Es el tipo de Panel que implementa nuestra View, por lo que se podría decir que nuestra StartView.java no es más que una clase que extiende de SplitLayoutPanel a la que se le han añadido diferentes hijos.

La estructura que sigue un SplitLayoutPanel se puede observar en la ilustración 24.

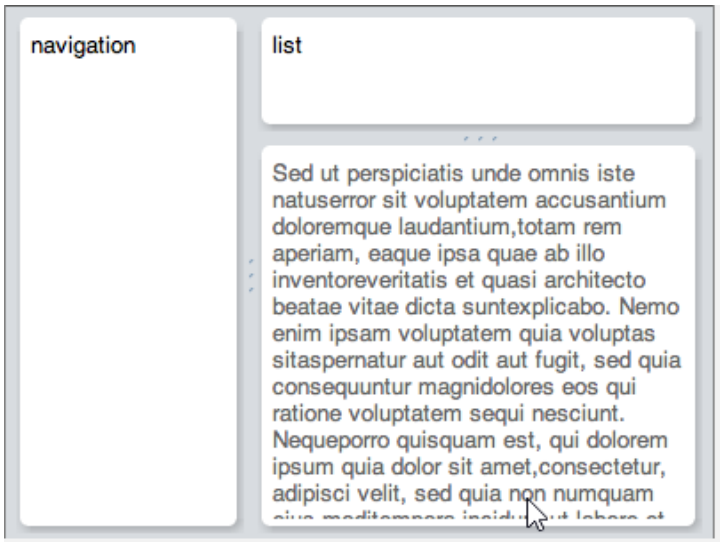



Ilustración 24 - SplitLayoutPanel

**NorthMenuBar**

Panel generado por nosotros mismos para representar la información superior de la vista principal.

Implementa un HorizontalPanel y está compuesto por Images y Labels.

**FooterBar**

Como en el caso anterior, se trata de una extensión de la clase HorizontalPanel en la que se disponen varias imágenes a modo de botones.

**VerticalPanel/HorizontalPanel**

**LateralPanel**

Extiende de VerticalPanel contiene elementos de tipo Image.

**ScrollPanel**

**LoginPanel**

Componente creado por nosotros y que usamos para mostrar el formulario de registro/login de usuarios. Es una extensión de un DialogBox y está compuesto por VertialPanels, HorizaontalPanels, Labels, TextBox y Buttons.

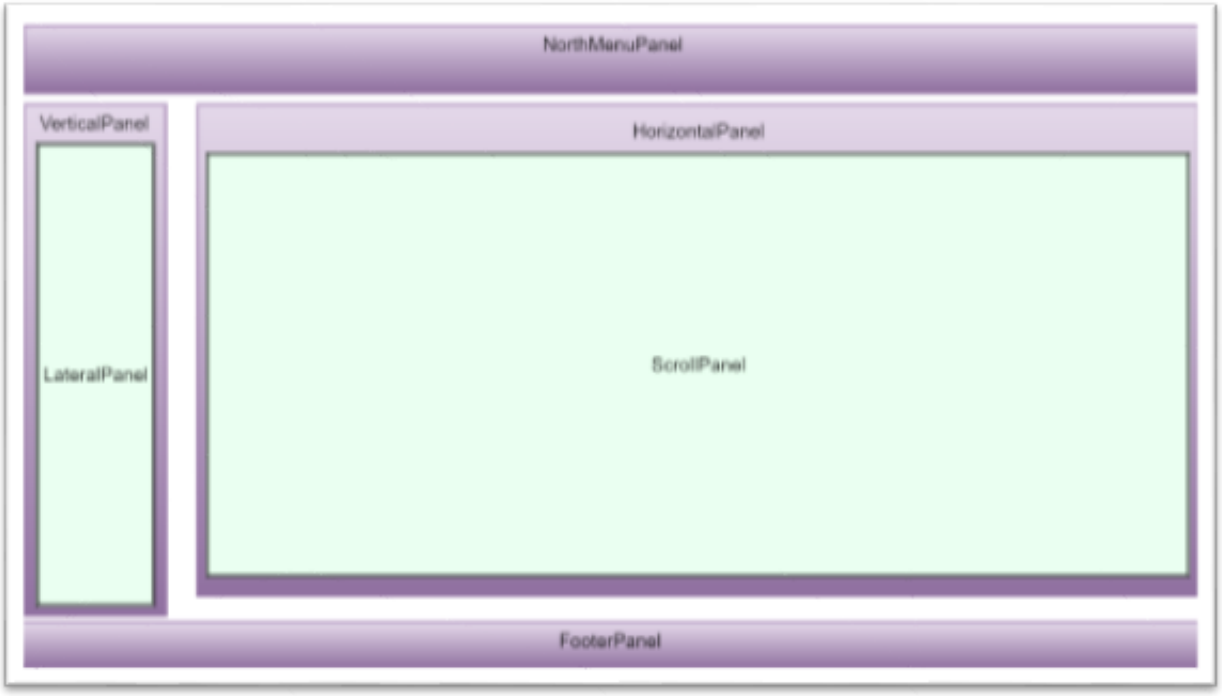



Ilustración 25 - Estructura StartView

- ***Construcción de la vista***

```
public class StartView extends SplitLayoutPanel implements StartPresenter.Display {
    private NorthMenuBar northMenuBar;
    private FooterBar footerPanel;
    private VerticalPanel lateralWrapper;
    private LateralPanel lateralPanel;
    private HorizontalPanel contentWrapper;
    private ScrollPanel contentPanel;
    private LoginPanel loginPanel;
    private Label label;

    public StartView() {
        super();
        setMainDockPanelParameters();
    }

    private void setMainDockPanelParameters() {
        Window.enableScrolling(false);
        Window.setMargin("0px");

        addChildrens();

    }
```

Ilustración 26 - Desarrollo StartView 1

Como bien hemos mencionado anteriormente, y como se puede observar en la ilustración 26, nuestra clase StartView extiende del Panel de GWT SplitLayoutPanel, a la vez que implementa la interfaz Display creada por StartPresenter.java.

```
private void addChildrens() {
    //Elemento menú superior
    northMenuBar = new NorthMenuBar();
    northMenuBar.setStyleName("northMenuBar");

    //Elemento menú inferior
    footerPanel = new FooterBar();
    footerPanel.setStyleName("footerPanel");

    //Elementos menú lateral
    lateralWrapper = new VerticalPanel();
    lateralWrapper.setStyleName("lateralWrapper");
    lateralWrapper.setVerticalAlignment(HasVerticalAlignment.ALIGN_MIDDLE);
    lateralPanel = new LateralPanel();
    lateralPanel.setStyleName("lateralPanel");

    //Elementos para el contenido
    contentWrapper = new HorizontalPanel();
    contentWrapper.setVerticalAlignment(HasVerticalAlignment.ALIGN_MIDDLE);
    contentWrapper.setStyleName("contentWrapper");

    contentPanel = new ScrollPanel();
    contentPanel.setStyleName("contentScrollPanel");

    //Elemento formulario de registro/login
    loginPanel = new LoginPanel();
    loginPanel.setStyleName("dialogBox");
    loginPanel.setVisible(false);

    // Image directamente vs dialogbox+image vs simplepanel+image
    FlowPanel fp = new FlowPanel();
    fp.setStyleName("contentScrollPanel");
    Image title = new Image("/images/title.png");
    title.setStyleName("title");
    fp.add(loginPanel);
    fp.add(title);
    contentPanel.add(fp);

    contentWrapper.add(contentPanel);
    lateralWrapper.add(lateralPanel);
```

**Ilustración 27 - Desarrollo StartView 2**

```
    Integer clientHeight = new Integer(Window.getClientHeight());
    Integer clientWidth = new Integer(Window.getClientWidth());

    double menuHeight = clientHeight.doubleValue() * 0.15;

    double footerHeight = clientHeight.doubleValue() * 0.10;
    double lateralMenuHeight = clientWidth.doubleValue() * 0.05;
    addNorth(northMenuBar, menuHeight);
    addSouth(footerPanel, footerHeight);
    addWest(lateralWrapper, lateralMenuHeight);
    add(contentWrapper);

}
```

Ilustración 28 - Desarrollo StartView 3

Se puede ver en el código anterior como se instancian e inicializan cada uno de los paneles descritos en el apartado *Panels empleados*.

Lo último que cabe destacar de la implementación de cada una de las View, es que al ser implementaciones de la interfaz Display de su Presenter correspondiente, se fuerza a que implementen los métodos descritos dentro de la interfaz.

Uno de los métodos descritos dentro de StartPresenter.Display era el método getLoginButton(). Pues bien, en la parte de la View, se implementará de la siguiente manera:

```
@Override
public HasClickHandlers getLoginButton() {
    return loginPanel.getLoginButton();
}
```

Ilustración 29 - Implementación método de Display

Lo único que hace este método es devolver un Widget de tipo Button perteneciente al LoginPanel, el cual además cumple con el requisito de HasClickable(), por lo que es un objeto válido para devolver.

#### *5.2.1.4 Llamadas RPC*

Una llamada RPC (Remote Procedure Call) se trata de un mecanismo mediante el cual dos procesos independientes se pueden comunicar entre sí.

Cada uno de los procesos mencionados pueden estar en redes diferentes, en Internet, o en la misma red y mismo ordenador.

En la parte cliente del proyecto GWT, tenemos dos clases relacionadas directamente con el servicio RPC: InGarduinoRPCInterface.java y InGarduinoRPCInterfaceAsync.java.

En ellas se describen, sin implementar, los métodos que vamos a querer tener disponibles para hacer la llamada RPC y recuperar da

```
@RemoteServiceRelativePath("ingarduinorpc")
public interface InGarduinoRPCInterface extends RemoteService {


    public User authenticateUser(String username, String password);
    public User loginFromServerSession();
    public void logout();
```

Ilustración 30 - Declaración síncrona de los métodos RPC

```
public interface InGarduinoRPCInterfaceAsync {


    public void authenticateUser(String username, String password, AsyncCallback<User> callback);
    public void loginFromServerSession(AsyncCallback<User> callback);
    public void logout(AsyncCallback<Void> callback);
```

Ilustración 31 - Declaración asíncrona de los métodos RPC.

#### *5.2.1.5 Comunicación cliente-nodos*

Ya que en el proyecto no hemos desarrollado e implantado un sistema de jardín inteligente, para realizar diferentes pruebas nos hemos puesto en contacto con un grupo de alumnos de la Escuela Técnica Superior de Ingeniería Informática (ETSINF) de la Universidad Politécnica de Valencia (UPV) cuyo Trabajo Final de Grado era implantar un sistema de estas características.

Más concretamente, han montado un sistema inteligente de macetas móviles las cuales disponen de sensores y actuadores mediante micro-controladores Arduino.

Puesto que InGarduino se ha desarrollado para ser lo más genérico posible, tanto un proyecto como otro trabajan juntos a la perfección.

Lo único que nos ha hecho falta ha sido que nos facilitaran la información necesaria de la red donde estarán configurados los diferentes nodos y las URL necesarias para realizar las operaciones REST necesarias para obtener la información de los sensores y de los actuadores.

Para llevar a cabo las operaciones pertinentes vamos a emplear dos tecnologías: REST y JSON.

- ***REST (Representational State Transfer)***

Se trata de un conjunto de reglas para el diseño de una arquitectura de red encargada de trasmitir datos específicos de un dominio sobre el protocolo HTTP.

REST no es un estándar ya que simplemente es una forma de crear una arquitectura de comunicación, pero sí que emplea tecnologías estándares como HTTP, URL, XML/HTML, Tipos MIME...

Dispone de varios métodos HTTP para la transferencia de datos por red, pero para nuestro caso sólo nos harán fala el método GET (Para obtener información) y el método POST (Para actualizar información);

- ***JSON (JavaScript Object Notation)***

Para el enviar datos de la forma que hemos descrito en la arquitectura REST, necesitamos un formato de datos que sea lo más universal posible y que tenga un gran rango de admisión.

JSON cumple con estos requisitos, además de su gran ligereza gracias a estar basado en la tecnología JavaScript.

Su estructura podría equipararse a la de un java.util.Map, pero la diferencia es que JSON encapsula toda la información en un objeto literal, lo que hace muy fácil su manejo y envío.

- ***Actualización de los valores de los sensores de los nodos agregados***

Una vez hemos descrito cómo vamos a transferir nuestra información necesaria por la red, vamos a explicar el código implementado con el ejemplo de las macetas inteligentes.

Antes de nada, tendremos que haber agregado los nodos mediante la interfaz gráfica pertinente, en la vista general de los nodos. Al agregar cada uno de los nodos, se guarda su URL necesaria para luego realizar la llamada al método GET del sistema REST.

El siguiente código pertenece a la clase NodesManagamentPresenter.java , que es el Presenter encargado de la vista general de los nodos. Cuando hagamos click en el botón de actualizar la información, se llevarán a cabo las siguientes tareas:

```
public void updateNodesData() {
    String URL = "";
    RequestBuilder builder;
    Request request;
    try {
        //Para cada uno de los nodos que dispongamos, actualizar sus valores
        for (Node n : nodeList.getNodeList()) {
            URL = n.getNetConfigurations().getURL();
            builder = new RequestBuilder(RequestBuilder.GET, URL);

            //Esta petición devolverá el JSON con los valores de los sensores
            request = builder.sendRequest(null, new RequestCallback() {

                @Override
                public void onResponseReceived(Request request, Response response) {
                    if(response.getStatusCode() == 200){
                        updateNodeSensors(n.getId(),response.getText());
                    }else{
                        Window.alert("La respuesta no ha devuelto codigo 200 de HTTP");
                    }
                }

                @Override
                public void onError(Request request, Throwable exception) {
                    Window.alert("Error al intentar recuperar el valor de los sensores del nodo "+n.getNodeType().getName());
                }
            });
        }
    } catch (Exception e) {
        throw new RuntimeException(e);
    }
}
```

**Ilustración 32 - Método GET para obtener los valores de los sensores**

El funcionamiento es el siguiente:

1. De la lista de nodos, obtenemos la URL de cada uno de ellos.
2. Creamos el objeto RequestBuilder indicando el método GET y la URL de cada nodo.
3. Realizamos la petición.
4. Con los datos recibidos, actualizamos los valores para el nodo del que he obtenido la URL.

Una vez recibidos los datos mediante el método GET, se parsea el resultado a un objeto JSON, para después acceder a cada una de sus *keys*, con lo que obtendremos el valor de cada uno de los sensores para después actualizarlos en base de datos.

```
protected void updateNodeSensors(Integer idNode,String text) {
    JSONParser parser = new JSONParser(text, null, false);
    Object obj = parser.parse();

    JSONObject jsonObject = (JSONObject) obj;

    /**A partir de aqui sería una configuración diferente para cada jardín inteligente,
     *  ya que hay que utilizar la key de cada uno de los sensores (Ajeno a nosotros)
     *
     *  Como ejemplo,con la info. que nos han facilitado los compañeros del sistema de macetas inteligentes,
     *  obtendremos los siguientes sensores
     *  **/

    String tempAire = jsonObject.get("temperatura_aire").toString();
    String tempSuelo = jsonObject.get("temperatura_suelo").toString();
    String luz1 = jsonObject.get("luz1").toString();
    String luz2 = jsonObject.get("luz2").toString();

    updateSensorsInDB(idNode,tempAire,tempSuelo,luz1,luz2);


}
```

Ilustración 33 - Parseo resultado GET a JSON

- ***Actuación sobre los diferentes actuadores de los nodos agregados***

Para realizar la acción de que actúen los actuadores del nodo, simplemente tendremos que cambiar los dos primeros puntos del caso anterior, ya que únicamente tendremos que cambiar el método GET por el POST, y cambiar la URL a la correspondiente del actuador.

### 5.2.2 Parte servidor

#### *5.2.2.1 Llamadas RPC*

En la parte servidor del proyecto será donde implementemos los métodos del servicio RPC, mediante la clase InGarduinoRPCInterfaceImpl.java.

Esta clase será la encargada de realizar las conexiones pertinentes con nuestra base de datos, y una vez establecidas dichas conexiones, se ocupará tanto de recuperar como de guardar los datos.

```java
public class InGarduinoRPCInterfaceImpl extends RemoteServiceServlet implements InGarduinoRPCInterface {

    private static final String NEW_USER_PREFIX = "CU-";
    private Connection conn;
    private String dbName = "ingarduino";
    private String dbUrlConn = "jdbc:mysql://127.0.0.1:3306/" + dbName + "?autoReconnect=true&useSSL=false";
    private String dbUserConn = "InGarduinoDB";
    private String dbPassConn = "garden2016garden";

    public InGarduinoRPCInterfaceImpl() throws Exception {
        super();
        try {
            Class.forName("com.mysql.jdbc.Driver").newInstance();
        } catch (Exception e) {
            throw new Exception(e);
        }
    }

    @Override
    public User authenticateUser(String username, String password) {
        User dbUser = null;

        try {

            if (username.startsWith(NEW_USER_PREFIX)) {
                dbUser = createUser(username, password);
            } else {
                dbUser = authenticate(username, password);
            }
            saveUserInSession(dbUser);
        } catch (Exception e) {
            dbUser = null;
        }
        return dbUser;
    }
```

**Ilustración 34 - Vista general InGarduinoRPCInterfaceImpl()**

- ***Método para dar a un usuario de alta***

```
@Override
public User createUser(String name, String password) throws Exception {
    PreparedStatement ps = null;
    try {
        conn = DriverManager.getConnection(dbUrlConn, dbUserConn, dbPassConn);
        String cryptPass = BCrypt.hashpw(password, BCrypt.gensalt());

        ps = conn.prepareStatement("INSERT into  USERS (NAME,PASSWORD) VALUES (?,?)");
        ps.setString(1, name.substring(3));
        ps.setString(2, cryptPass);
        ps.execute();

    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        ps.close();
        conn.close();
    }
    return authenticate(name.substring(3), password);
}
```

Ilustración 35 - Método para registrar nuevo usuario

- ***Método que guarda al usuario autenticado en sesión***

```
private void saveUserInSession(User dbUser) {
    HttpServletRequest httpServletRequest = this.getThreadLocalRequest();
    HttpSession session = httpServletRequest.getSession(true);
    session.setAttribute("user", dbUser);
}
```

Ilustración 36 - Método para guardar el usuario en sesión

- ***Método para loguear usuario***

```
private User authenticate(String username, String password) throws Exception {
    User authenticatedUser = null;
    ResultSet result = null;
    PreparedStatement ps = null;
    Boolean authenticated = false;
    try {
        if (conn == null || conn.isClosed()) {
            conn = DriverManager.getConnection(dbUrlConn, dbUserConn, dbPassConn);
        }

        ps = conn.prepareStatement("SELECT * " + "FROM USERS WHERE NAME LIKE '" + username + "'");
        result = ps.executeQuery();

        while (result.next()) {
            authenticatedUser = new User(result.getInt(1), result.getString(2), result.getString(3));
        }

        // Se comprueba si el pass introducido corresponde con el de bbdd
        if (authenticatedUser != null) {
            authenticated = BCrypt.checkpw(password, authenticatedUser.getPassword());
        }

    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        result.close();
        ps.close();
        conn.close();
    }

    return authenticated ? authenticatedUser : null;

}
```

Ilustración 37 - Método para loguear usuario

## 5.3 Implantación

Para la implantación del servicio web vamos a emplear el contenedor de servlets Apache Tomcat, debido a su facilidad de instalación y de implantación de servicios en él.

Además, al haber sido programado en Java, el único requisito que necesita el dispositivo donde se instale el Tomcat es que disponga de una máquina virtual Java, casualmente, el único requisito que necesita también el servicio web para poder ser ejecutado.

Con la utiización de este software seguimos con nuestra línea de universalidad y generalización, ya que en la acutalidad la mayoría de dispositivos permiten Java, como por ejemplo los diferentes tipos de micro-controladores (Arduino, Raspberry...), que suelen ser los dispositivos que actúan como servidor en los proyectos relacionados con el Internet de las cosas.

El primer paso que debemos realizar para implantar el proyecto en el Tomcat es compilarlo para que así nos genere un archivo .war.

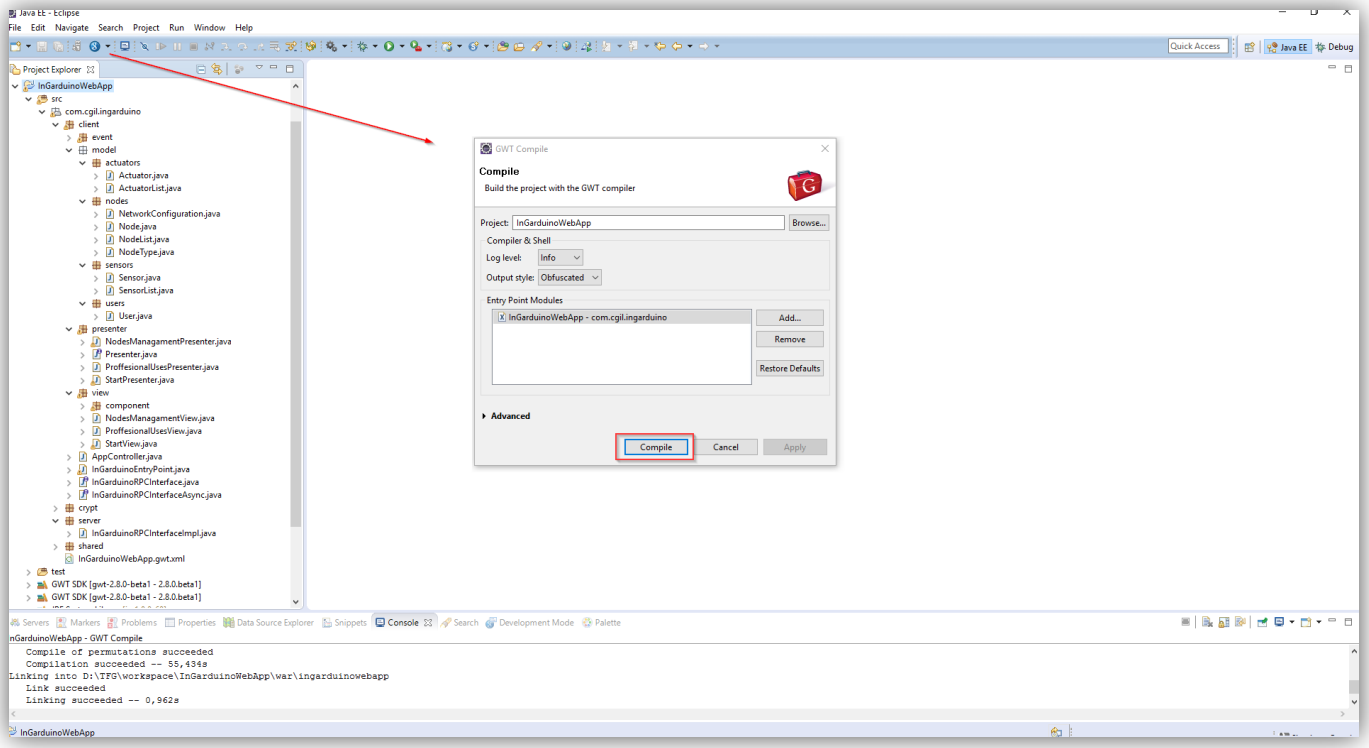



Ilustración 38 - Compilación proyecto GWT

Una vez nos ha compilado el servicio web, podemos observar en la ruta del proyecto que se nos ha generado una carpeta llamada "war", la cual, mediante cualquier programa de compresión/descompresión de archivos, comprimiremos en un archivo llamado InGarduino.war.

Es muy importante que la extensión del archivo sea .war y no cualquier otra oculta.

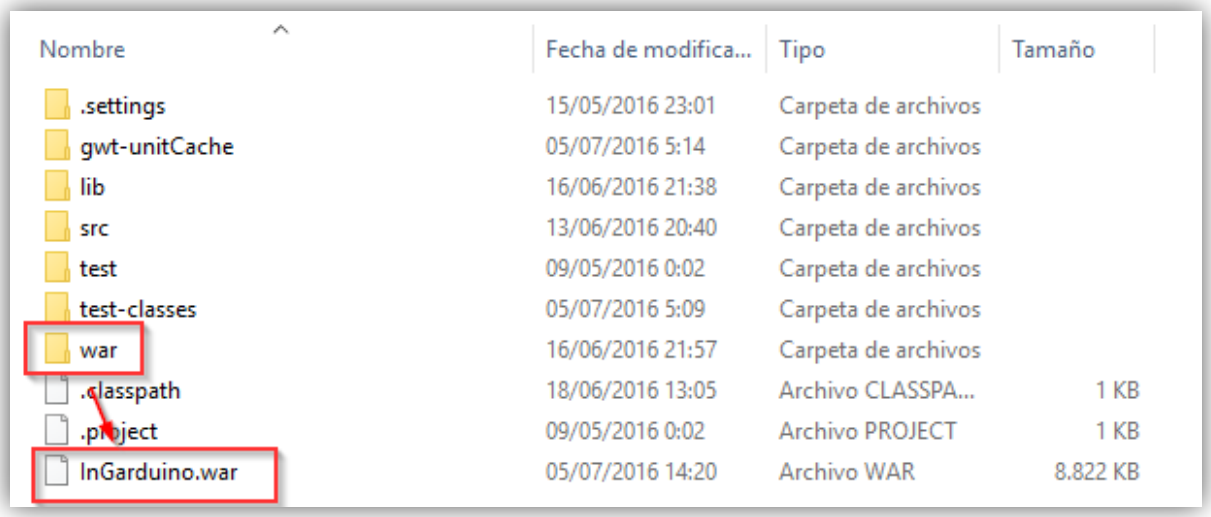
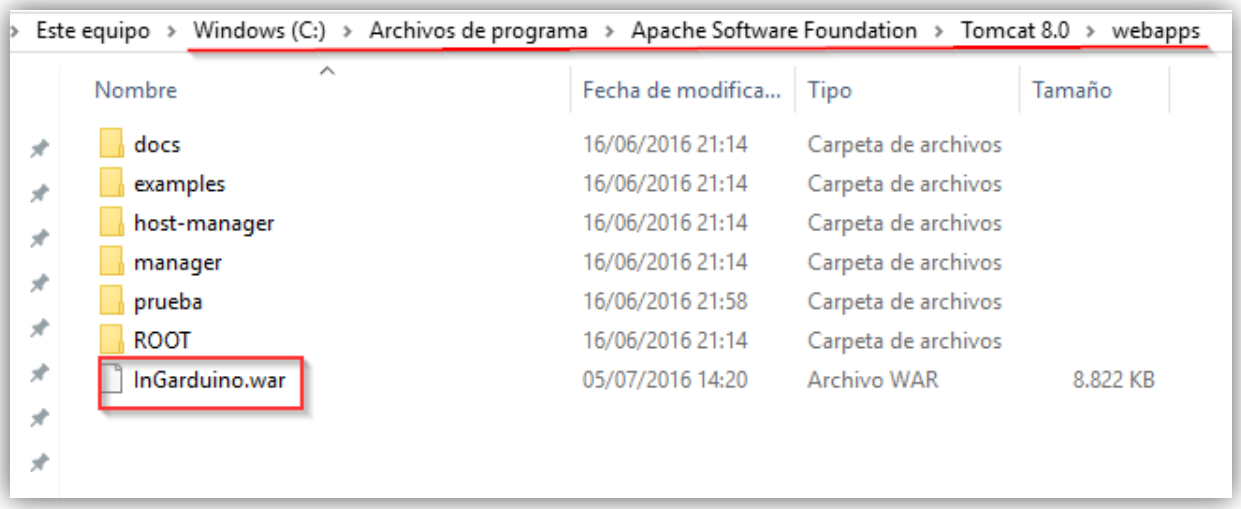



Ilustración 39 - Generación InGarduino.war

Por último, copiaremos este archivo InGarduino.war y lo pegaremos en la carpeta "webapps" del servidor Tomcat, que está en su raíz.




Finalmente, reiniciaremos el servicio del servidor Tomcat, y él automáticamente se encargará de desplegar los .war reconocidos en su carpeta de despliegue de aplicaciones.

Para asegurarnos completamente de que todo ha funcionado correctamente, tenemos que observar que en la misma carpeta donde habíamos dejado el archivo InGarduino.war, se ha generado una carpeta conel mismo nombre pero sin la extensión.

Una vez hechos todos estos pasos, el servicio web ya estará disponible para acceder a él desde cualquier navegador, accediendo a la siguiente URL:

IP_DEL_SERVIDOR:8080/InGarduino

## 5.4 Evaluación

A continuación vamos a ejecutar el servicio web InGarduino en el servidor Tomcat y vamos a comprobar que cumpla con todos los requisitos expuestos a lo largo de esta memoria.

- ***Pantalla de registro del usuario***

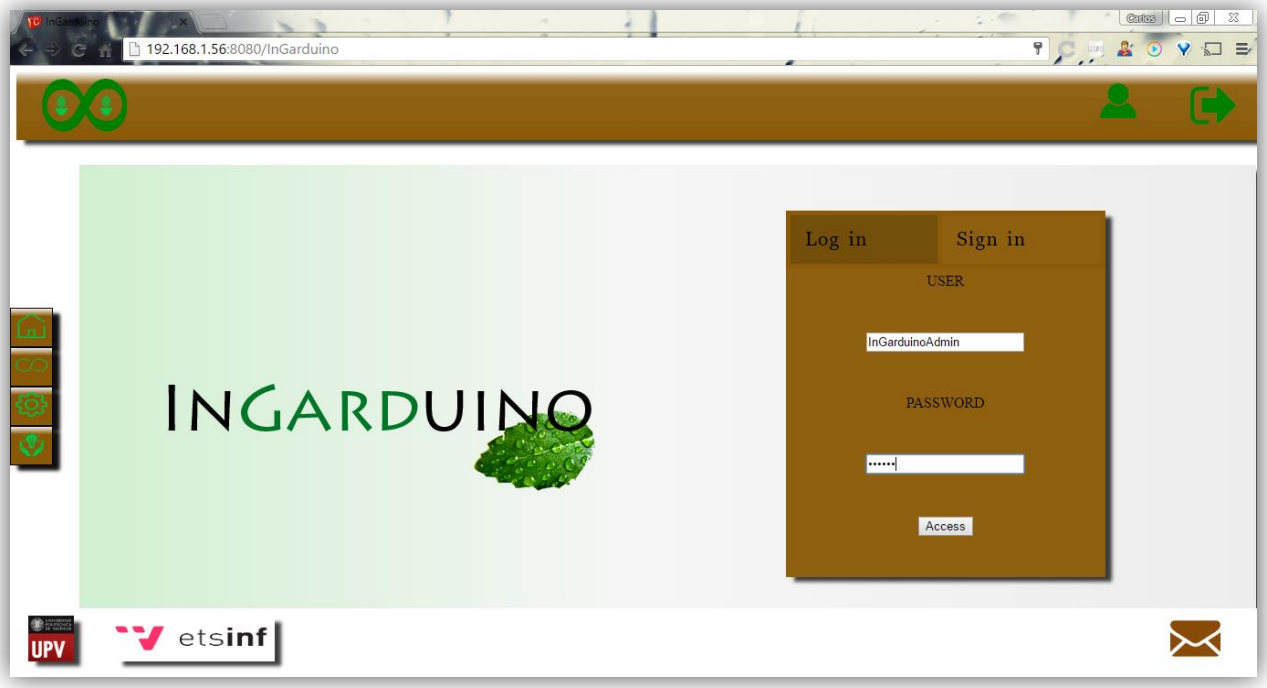



Ilustración 40 - Pantalla sign in

En la vista principal podemos elegir entre darnos de alta en el sistema, o loguearnos si ya nos hemos dado de alta con anterioridad.

Si se selecciona la pestaña de Sign in, crea un usuario nuevo con los datos del formulario y te inicia sesión automáticamente.

Si se selecciona la pestaña Log in, se inicia la sesión del usuario.

- ***Pantalla de acceso del usuario***

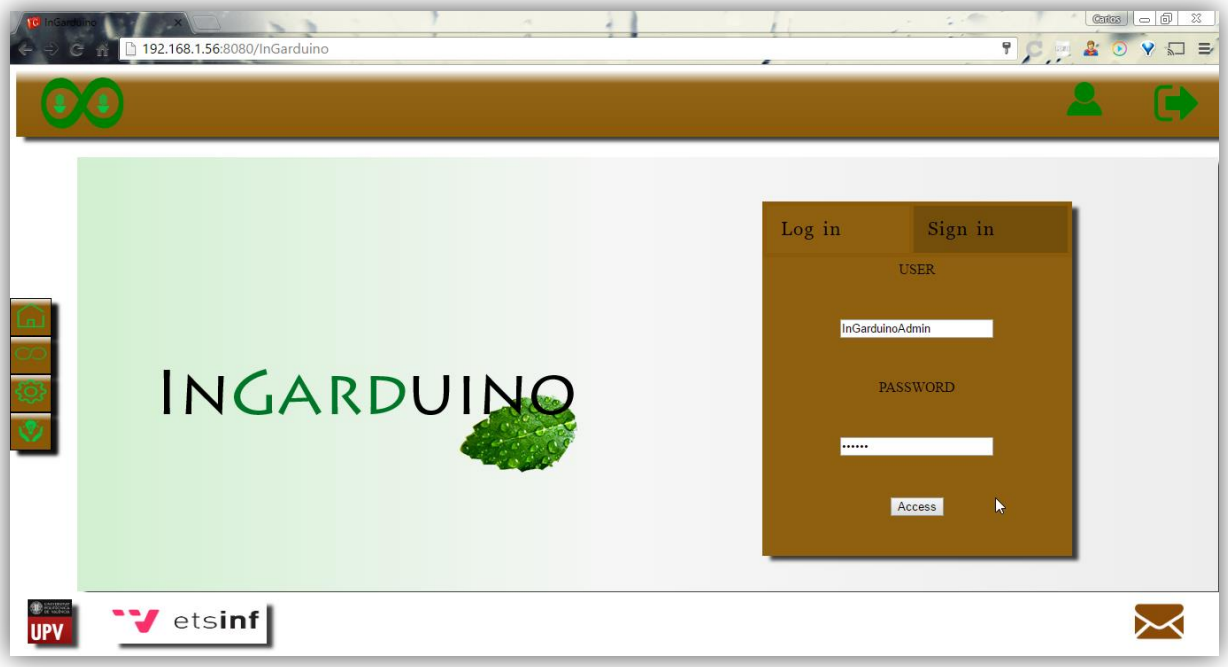



Ilustración 41 - Pantalla log in

- ***Pantalla de usuario logueado***


Ilustración 42 - Usuario logueado

Apenas se aprecia la diferencia entre las anteriores pantallas debido al tamaño de este documento, pero en el caso del usuario logueado se puede observar como aparece el nombre de usuario debajo del símbolo de Perfil.

- ***Pantalla de la vista principal de los nodos***

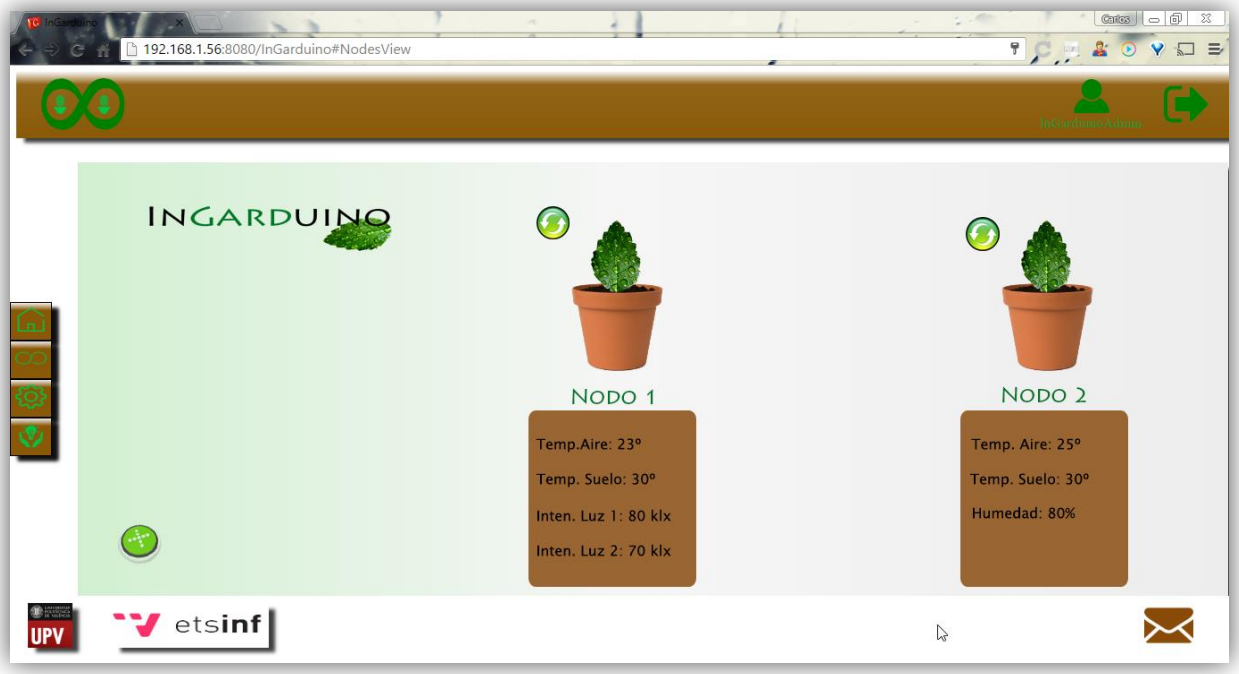



Ilustración 43 - Pantalla vista de los nodos

# 6 Conclusiones

El objetivo de este proyecto era diseñar y desarrollar una aplicación web capaz de poder gestionar la información que nos ofrece cualquier sistema de jardín inteligente, adentrándonos así un poco en ese mundo tan amplio del Internet de las cosas, cuyo objetivo es similar al nuestro, llevar el control de todo dispositivo que sea capaz de recibir y/o emitir información.

Debido a la coincidencia temporal entre el boom de las aplicaciones móviles y el del Internet de las cosas, en el análisis de mercado existente se pudo observar que existen muchas apps para smartphones y dispositivos móviles. En cambio, aplicaciones web específicas para controlar un jardín inteligente desde el ordenador apenas existen, y de las que existen, una grandísima mayoría están sujetas a funcionar sólo para el sistema inteligente que te implanta la misma empresa que la aplicación web.

Esto podría parecer una piedra en el camino por no saber hacia dónde tirar a la hora de qué tendrá que hacer o cómo lo tendrá que hacer, pero nada parecido. Esta falta de mercado nos ha servido para no haber estado influenciados en la forma en la que desarrollar el servicio web, permitiéndonos un grado de libertad que no se puede conseguir en el mercado de las apps móviles que parece que si intentas hacer algo fuera de la app plantilla, está mal.

Pero no es todo malo el mundo de las apps móviles, su sencillez a la hora de usar la interfaz gráfica y su universalidad nos ha servido para coger muchas ideas.

Con este objetivo en mente de hacer el servicio lo más liviano, sencillo y general posible, elegimos para desarrollar el lenguaje Java mediante el framework Google Web Toolkit, lo que ha sido todo un acierto. No sólo ya por el ahorro de tiempo que GWT con su hosted mode que te da al programar, actualizar y ver resultado al instante, sino sobre todo la multiplataforma que nos permite Java.

Fue difícil configurar todo el entorno de programación y que funcionara todo correctamente, pero en cuanto se consigue, trabajar con este framework se hace muy ágil.

Finalmente, se puede decir que se ha alcanzado el objetivo del proyecto, ya que tras el análisis, diseño, desarrollo e implementación, se ha conseguido un servicio web sencillo pero que a su vez que cumple con todos los requisitos necesarios para una gestión fácil de cualquier jardín inteligente

# 7 Referencias

[1]Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. Future Generation Computer Systems, 29(7), 1645-1660.

[2]Premeaux, E., Evans, B., & Turner, M. (2011). Arduino Projects to Save the World. Apress.

[3]Richardson, M., & Wallace, S. (2012). Getting started with raspberry PI. " O'Reilly Media, Inc.".

[4]Shiflett, C. (2003). HTTP Developer's Handbook. Sams Publishing.

[5]Masse, M. (2011). REST API design rulebook. " O'Reilly Media, Inc.".

[6]Bloch, J. (2008). Effective java. Pearson Education India.

[7]Dwyer, J. (2008). Pro Web 2.0 Application Development with GWT. Apress.

[8]Smink, J. (2004). Ajax Training Sessions. Reedswain Inc..

[9]Sheldon, R., & Moes, G. (2005). Beginning MySQL. John Wiley & Sons.

[10]Burnette, E. (2005). Eclipse IDE Pocket Guide. " O'Reilly Media, Inc.".

[11]Duckett, J. (2008). Beginning Web programming with HTML, XHTML, and CSS. John Wiley & Sons.

[12]Flanagan, D. (2006). JavaScript: The Definitive Guide: The Definitive Guide. " O'Reilly Media, Inc.".

[13]Potel, M. (1996). MVP: Model-View-Presenter the Taligent programming model for C++ and Java. Taligent Inc, 20.