

**POOL  
DUINO**

# DESARROLLO DE UNA APLICACIÓN WEB PARA EL CONTROL Y LA MONITORIZACIÓN DE UNA PISCINA

MEMORIA DEL TRABAJO FINAL DE GRADO

AUTOR

**SERGIO LEÓN ESQUIVEL**

CONSULTOR

**KENNETH CAPSETA NIETO**

PROFESOR

**CARLOS CASADO MARTÍNEZ**

GRADO MULTIMEDIA

DISEÑO DE APLICACIONES INTERACTIVAS

24 DE JUNIO DEL 2014

 **UOC**

Universitat Oberta  
de Catalunya

## Créditos y derechos

El autor del proyecto es **Sergio León Esquivel**. Este trabajo está sujeto a una licencia *Creative Commons* de tipo Reconocimiento – NoComercial – SinObraDerivada (*by-nc-nd: Attribution – NonCommercial – NoDerivateWorks*).



Figura 1. Licencia *Creative Commons by-nc-nd*

Se puede acceder a una copia de la licencia a través de la página:

<http://creativecommons.org/licenses/by-nc-nd/4.0/>

El proyecto hace uso de librerías, *software* y recursos que son propiedad de terceros:

Apache Cordova (<http://cordova.apache.org/>)

Arduino (<http://www.arduino.cc/>)

Bootstrap (<http://getbootstrap.com/>)

Bootstrap DatePicker (<http://www.eyecon.ro/bootstrap-datepicker/>)

Bootstrap TimePicker (<http://jdewit.github.io/bootstrap-timepicker/>)

Flot (<http://www.flotcharts.org/>)

FontAwesome (<http://fontawesome.io/>)

jQuery (<http://jquery.com/>)

jQuery UI (<http://jqueryui.com/>)

MySQL (<http://www.mysql.com/>)

PhoneGap (<http://phonegap.com/>)

PHP (<http://www.php.net/>)

PHPMailer (<http://phpmailer.worxware.com/>)

Slim (<http://www.slimframework.com/>)

TouchSwipe (<http://labs.rampinteractive.co.uk/touchSwipe/demos/>)

## Agradecimientos

*A mi pareja, por caminar a mi lado y darme la fuerza necesaria para seguir adelante en los momentos más difíciles. Por hacerme ver que soy capaz de todo, por acompañarme en todas las noches que he tenido que quedarme despierto, y aún así no dejar de regalarme una sonrisa repleta de ternura.*

*A mi padre y a mi madre, por darlo todo por mí sin esperar nada a cambio y por anteponer mis necesidades a las suyas. Por quererme y apoyarme ciegamente en todo lo que he hecho, hago y haré. Por ser un pilar fundamental para mí.*

*A mis tres hermanos, por su soporte incondicional a lo largo no sólo de la carrera sino de toda mi vida, por dejarlo todo para ayudarme siempre que lo he necesitado. Por ser una referencia de lo que quiero llegar a ser en el futuro.*

*A mi cuñado y a mis dos cuñadas, por ser una parte indispensable de mi vida y por seguirme tanto en mis derrotas como en mis victorias.*

*A mis sobrinos, porque sólo con ver sus pequeñas caras de felicidad tengo fuerzas para mover montañas.*

*A mis suegros, porque media carrera la he pasado en su casa y aún así no me han echado. Por soportarme y ayudarme, pero sobre todo, por hacerme uno de los mejores regalos que podría haber imaginado.*

*A mis amigos, muy responsables de hacerme ser quien soy hoy. Por acompañarme tantos años y aún así hacerme mejorar día a día.*

*Y a todo aquel que a lo largo de mi vida me ha enseñado que las barreras nos las ponemos nosotros mismos.*

*Simplemente, gracias por todo.*

## Abstract

El mantenimiento de una piscina privada es un procedimiento que implica cierta complejidad y requiere unos conocimientos técnicos básicos. Por regla general, dicho proceso se lleva a cabo de forma manual.

Este proyecto trata el desarrollo de una página web que permite recopilar datos de una piscina en tiempo real para su monitorización en un navegador, de forma que el usuario puede conocer la situación actual del sistema a través de Internet. Asimismo, la plataforma permite la interacción con la piscina de forma remota para poder actuar sobre el *hardware* sin estar físicamente presente.

Se desarrolla también una versión de la página como aplicación para los principales dispositivos móviles, de forma que el usuario puede instalarla en su terminal y recibir notificaciones ante la aparición de ciertos sucesos que requieran su intervención manual.

Palabras clave: UOC, Trabajo Final de Grado, Desarrollo aplicaciones interactivas, Multimedia, Arduino, Internet de las cosas, Aplicación web, Piscina, Dispositivo móvil, PoolDuino

*Maintenance of a private pool is a procedure that involves some complexity and requires some basic technical knowledge. Generally, this process is carried out manually.*

*This project covers the development of a web page which allows the user to collect pool's data for real time monitoring in a browser, so he can check the current status of the system through Internet. Additionally, the platform allows him to remotely interact with the pool so it is possible to act on the hardware without being physically present.*

*A version of the page as an application for main mobile devices is developed as well, so the user can install it in his terminal and receive further notifications when certain events which may require manual intervention occur.*

*Keywords: UOC, Undergraduate Thesis Project, Interactive application development, Multimedia, Arduino, Internet of Things, Web Application, Pool, Mobile device, PoolDuino*

## Notaciones y convenciones

A continuación se muestra un listado de los estilos tipográficos que se encontrarán a lo largo de la memoria para poder distinguir su función dentro del documento.

### Título de primer nivel

### Título de segundo nivel

### *Título de tercer nivel*

Texto normal

**Texto resaltado**

*Términos técnicos o palabras en otro idioma*

```
1 | Código de programación
2 | comentarios
3 | keywords o palabras clave
```

**Código/Figura/Tabla Núm.** Pie de código/figura/tabla

- Primer punto de lista sin orden
  - Segundo punto de lista sin orden
- 1 Primer punto de lista con orden
  - 2 Segundo punto de lista con orden

<i>Columna 1</i>	<i>Columna 2</i>	<i>Columna 3</i>
Celda 1 Fila 1	Celda 2 Fila 1	Celda 3 Fila 1
Celda 1 Fila 2	Celda 2 Fila 2	Celda 3 Fila 2
Celda 1 Fila 3	Celda 2 Fila 3	Celda 3 Fila 3

# Índice

1. Introducción/Prefacio .....	11
2. Descripción/Definición/Hipótesis .....	12
3. Objetivos .....	14
3.1. Principales .....	14
3.2. Secundarios .....	14
4. Escenario .....	15
5. Contenidos.....	16
6. Metodología .....	17
7. Arquitectura de la plataforma.....	18
7.1. Cliente .....	18
7.2. Servidor .....	18
7.3. Base de datos .....	19
7.4. Arduino .....	21
8. Plataforma de desarrollo .....	22
8.1. Software .....	22
8.2. Hardware .....	22
8.3. Otros servicios.....	23
9. Planificación.....	24
9.1. Fechas clave .....	24
9.2. Hitos .....	24
9.3. Diagrama de Gantt .....	25
10. Proceso de desarrollo .....	26
10.1. Diseño inicial .....	26
10.2. Conexión con Arduino .....	26
10.3. Implementación del servidor y de la base de datos .....	29
10.4. Comunicación entre Arduino y el servidor .....	30
10.5. Diseño de la interfaz y el logotipo .....	30
10.6. Desarrollo de la interfaz de usuario .....	31
10.7. Desarrollo del sistema de notificaciones .....	32
10.8. Detección y corrección de errores .....	33
11. APIs utilizadas .....	34
11.1. API Arduino .....	34

11.2.	API Servidor.....	35
11.3.	SDK Android .....	36
11.4.	API PhoneGap.....	36
12.	Diagramas UML.....	37
12.1.	Arduino .....	37
12.2.	Servidor .....	37
13.	Prototipos .....	38
13.1.	Conexión con Arduino .....	38
13.2.	<i>Wireframes</i> de baja fidelidad .....	39
13.3.	<i>Wireframes</i> de alta fidelidad .....	40
13.4.	Logotipo .....	42
13.5.	Página de descarga .....	42
14.	Perfiles de usuario .....	43
15.	Usabilidad .....	44
16.	Seguridad.....	45
17.	Juegos de pruebas.....	47
17.1.	Evaluación técnica .....	47
17.2.	Evaluación de funcionalidad .....	47
17.3.	Evaluación con usuarios.....	47
18.	Versiones de la plataforma.....	48
18.1.	Versión Alpha .....	48
18.2.	Versión Beta.....	48
18.3.	Versión 1.0 - Release .....	49
19.	Requisitos de instalación.....	50
19.1.	Cliente desde navegador .....	50
19.2.	Cliente desde dispositivo móvil .....	50
19.3.	Servidor .....	50
19.4.	Arduino .....	50
20.	Instrucciones de instalación .....	51
21.	Instrucciones de uso .....	53
22.	Incidencias.....	54
23.	Proyección a futuro .....	56
24.	Presupuesto .....	58
25.	Conclusiones .....	59

## Índice de anexos

Anexo 1: Entregables del proyecto .....	61
Anexo 2: Código fuente .....	63
Anexo 3: Librerías/código externo utilizado .....	71
Anexo 4: Capturas de pantalla.....	75
Anexo 5: Guía de usuario .....	78
Anexo 6: Libro de estilo .....	83
Anexo 7: Glosario .....	84
Anexo 8: Bibliografía .....	87
Anexo 9: Vita .....	89

## Índice de figuras

<b>Figura 1.</b> Licencia <i>Creative Commons by-nc-sa</i> .....	2
<b>Figura 2.</b> Tabla de usuarios .....	19
<b>Figura 3.</b> Tabla de dispositivos asociados al usuario .....	19
<b>Figura 4.</b> Tabla de dispositivos .....	19
<b>Figura 5.</b> Tabla de métodos disponibles para el dispositivo .....	19
<b>Figura 6.</b> Tabla de parámetros para el método del dispositivo.....	19
<b>Figura 7.</b> Tabla de tipos de parámetros .....	19
<b>Figura 8.</b> Tabla de canales.....	20
<b>Figura 9.</b> Tabla de etiquetas para los valores de un canal .....	20
<b>Figura 10.</b> Tabla de <i>data points</i> o mediciones .....	20
<b>Figura 11.</b> Tabla de disparadores a nivel de canal .....	20
<b>Figura 12.</b> Tablas de disparadores a nivel de dispositivo.....	20
<b>Figura 13.</b> Tabla de dispositivos móviles.....	20
<b>Figura 14.</b> Tabla de notificaciones enviadas .....	20
<b>Figura 15.</b> Tabla de roles de acceso .....	20
<b>Figura 16.</b> Diagrama de Gantt del proyecto .....	25
<b>Figura 17.</b> Ejemplo de notificación .....	33
<b>Figura 18.</b> Prototipo: Pantalla de <i>login</i> .....	38
<b>Figura 19.</b> Prototipo: Usuario administrador.....	38
<b>Figura 20.</b> Prototipo: Usuario consultor .....	38
<b>Figura 21.</b> Prototipo: Datos de la solicitud.....	39
<b>Figura 22.</b> Prototipo: Respuesta de la solicitud.....	39



<b>Figura 23.</b> <i>Wireframe</i> baja fidelidad: <i>Login</i> .....	39
<b>Figura 24.</b> <i>Wireframe</i> baja fidelidad: <i>Login</i> – Campos vacíos .....	39
<b>Figura 25.</b> <i>Wireframe</i> baja fidelidad: <i>Login</i> – Error de credenciales .....	39
<b>Figura 26.</b> <i>Wireframe</i> baja fidelidad: Bloqueo de sesión .....	39
<b>Figura 27.</b> <i>Wireframe</i> baja fidelidad: <i>Dashboard</i> .....	40
<b>Figura 28.</b> <i>Wireframe</i> baja fidelidad: <i>Dashboard</i> – Sistema caído .....	40
<b>Figura 29.</b> <i>Wireframe</i> baja fidelidad: Canal .....	40
<b>Figura 30.</b> <i>Wireframe</i> baja fidelidad: Canal – Selección de intervalo .....	40
<b>Figura 31.</b> <i>Wireframe</i> baja fidelidad: Acciones .....	40
<b>Figura 32.</b> <i>Wireframe</i> baja fidelidad: Acciones – Ejecución .....	40
<b>Figura 33.</b> <i>Wireframe</i> baja fidelidad: Notificaciones .....	40
<b>Figura 34.</b> <i>Wireframe</i> baja fidelidad: Acciones de sesión .....	40
<b>Figura 35.</b> <i>Wireframe</i> alta fidelidad: <i>Login</i> .....	41
<b>Figura 36.</b> <i>Wireframe</i> alta fidelidad: <i>Login</i> – Campos vacíos .....	41
<b>Figura 37.</b> <i>Wireframe</i> alta fidelidad: <i>Login</i> – Error de credenciales .....	41
<b>Figura 38.</b> <i>Wireframe</i> alta fidelidad: Bloqueo de sesión .....	41
<b>Figura 39.</b> <i>Wireframe</i> alta fidelidad: <i>Dashboard</i> .....	41
<b>Figura 40.</b> <i>Wireframe</i> alta fidelidad: <i>Dashboard</i> – Sistema caído .....	41
<b>Figura 41.</b> <i>Wireframe</i> alta fidelidad: Canal .....	41
<b>Figura 42.</b> <i>Wireframe</i> alta fidelidad: Canal – Selección de intervalo .....	41
<b>Figura 43.</b> <i>Wireframe</i> alta fidelidad: Acciones .....	41
<b>Figura 44.</b> <i>Wireframe</i> alta fidelidad: Acciones – Ejecución .....	41
<b>Figura 45.</b> <i>Wireframe</i> alta fidelidad: Notificaciones .....	41
<b>Figura 46.</b> <i>Wireframe</i> alta fidelidad: Acciones de sesión .....	41
<b>Figura 47.</b> Logotipo .....	42
<b>Figura 48.</b> Página de descarga .....	42
<b>Figura 49.</b> Instalación: Casilla de orígenes desconocidos .....	51
<b>Figura 50.</b> Instalación: Confirmación de descarga .....	51
<b>Figura 51.</b> Instalación: Permisos .....	52
<b>Figura 52.</b> Instalación: Icono de la aplicación .....	52
<b>Figura 53.</b> Incidencias: <i>Script</i> sin respuesta .....	54
<b>Figura 54.</b> Arduino IDE accediendo vía escritorio remoto .....	75
<b>Figura 55.</b> Acceso a Arduino mediante HTTP y REST .....	76
<b>Figura 56.</b> Acceso a Arduino mediante el prototipo creado en la fase 1 .....	76

<b>Figura 57.</b> Información dinámica del canal ‘Temperatura exterior’ .....	77
<b>Figura 58.</b> Guía de usuario: <i>Login</i> y Estructura de la aplicación .....	78
<b>Figura 59.</b> Guía de usuario: Usuario y <i>Dashboard</i> .....	79
<b>Figura 60.</b> Guía de usuario: Detalle del canal .....	80
<b>Figura 61.</b> Guía de usuario: Acciones y Notificaciones .....	81
<b>Figura 63.</b> Guía de usuario: Notificaciones y Definición de reglas .....	82
<b>Figura 62.</b> Libro de estilo .....	83

## Índice de fragmentos de código

<b>Código 1.</b> Función <i>adjustFont</i> para ajuste automático de tamaño de fuente – app.js.....	54
<b>Código 2.</b> Configuración de la dirección IP y MAC del dispositivo – pool.ino.....	63
<b>Código 3.</b> Configuración de la <i>Ethernet shield</i> – pool.ino .....	63
<b>Código 4.</b> Inicialización de las clases API y RESTAPI – pool.ino .....	63
<b>Código 5.</b> Creación del servidor web – pool.ino .....	64
<b>Código 6.</b> Configuración del servidor web – pool.ino.....	64
<b>Código 7.</b> Función de tratamiento de solicitudes REST – pool.ino.....	64
<b>Código 8.</b> Envío de datos: Primera conexión – API.cpp .....	65
<b>Código 9.</b> Envío de datos: Envío periódico – API.cpp.....	65
<b>Código 10.</b> Envío de datos: Generación de estructura JSON – API.cpp .....	66
<b>Código 11.</b> Envío de datos: Comunicación con el servidor – API.cpp.....	66
<b>Código 12.</b> Código de comunicación con Arduino – PoolDuinoAPI.Methods.php .....	67
<b>Código 13.</b> Código de control de solicitud REST API – route.php.....	68
<b>Código 14.</b> Fragmento de PoolDuinoAPIException – PoolDuinoAPIException.php.....	68
<b>Código 15.</b> Uso de PoolDuinoAPIException – PoolDuinoAPI.Devices.php .....	68
<b>Código 16.</b> Envío de notificaciones <i>push</i> – PoolDuinoAPI.Notifications.php.....	69
<b>Código 17.</b> Envío de notificaciones por <i>email</i> – PoolDuinoAPI.Notifications.php .....	70
<b>Código 18.</b> Acceso a información del dispositivo y registro para GCM – app.js.....	70
<b>Código 19.</b> Método WebServer::dispatchCommand – WebServer.cpp .....	72

## Índice de tablas

<b>Tabla 1.</b> Fechas clave del proyecto.....	24
<b>Tabla 2.</b> Hitos parciales del proyecto .....	25
<b>Tabla 3.</b> Puntos de entrada de la API de Arduino .....	35
<b>Tabla 4.</b> Puntos de entrada de la API de Servidor .....	36

# 1.Introducción/Prefacio

La idea del proyecto nace a raíz de la necesidad de poder controlar una piscina privada de forma que requiera menos esfuerzo por parte del usuario, reduciendo así la complejidad del proceso y proporcionándole una plataforma que permita abstraerle del procedimiento interno que ejecuta el sistema para que todos los elementos funcionen de forma adecuada.

Este trabajo aúna la pasión del autor por el desarrollo de software y el diseño gráfico con una relativamente nueva afición al prototipado de sistemas mediante controladores Arduino, que ofrecen un sinfín de posibilidades. Con él pretende dar respuesta a las necesidades concretas de su hermano mayor, propietario de la piscina para la que se realizará la aplicación web de monitorización.

El proyecto actual parte de un trabajo previo en el que participó activamente tanto en su definición como en su consecución. En él se creó un sistema mediante una placa Arduino que era capaz de controlar los elementos físicos de la piscina permitiendo su interacción con ellos mediante pulsadores, a la vez que se instalaron diversos sensores que permitían obtener medidas *in situ* de las diferentes variables a tener en cuenta para su mantenimiento. Así pues, este trabajo se entiende como una extensión de un proyecto personal anterior.

Se podría definir como un interesante trabajo de **domótica** que tiene muchas posibles aplicaciones posteriores, ya que de abrir el número de dispositivos a controlar se estaría desarrollando una plataforma de gestión remota de dispositivos electrónicos propia del *Internet of Things* que tanta relevancia está ganando últimamente.

Este proyecto supone un reto educativo, personal y profesional. Educativo, porque es el trabajo de mayor envergadura al que se enfrenta a lo largo del grado. Personal, porque supone un proyecto que debe ser capaz de realizar íntegramente desde su concepción hasta su consecución, solventando todos los problemas que puedan darse en el proceso. Y profesional, porque existe la intención de extender la plataforma creada con el fin de introducirla posteriormente en el mercado.

## 2. Descripción/Definición/Hipótesis

El proyecto que se expone en esta memoria trata todo el proceso de desarrollo de una aplicación web para gestionar una piscina de forma remota mediante un *smartphone* o cualquier otro terminal, aunque sin limitarlo a ese soporte ya que realmente sólo se trata de una página web con diseño responsivo. No obstante, hace falta remarcar la importancia del diseño específico para este dispositivo, ya que su uso primordial es a través de un terminal móvil para poder consultar el estado actual de la piscina y actuar en consecuencia cuando el usuario no está en casa.

Dicho proyecto forma parte de la mención de diseño de aplicaciones interactivas y engloba todas las fases del proceso –análisis de requerimientos, conceptualización, desarrollo, diseño y evaluación del sistema generado–.

*Grosso modo*, el usuario requiere gestionar los datos recopilados por un dispositivo electrónico de la forma más sencilla e intuitiva posible. Para dar solución a sus necesidades se extiende el sistema Arduino actual para incorporar un módulo *Ethernet* con el que poder enviar los datos recopilados por la placa a un servidor externo que procesa toda la información enviada, almacenándola en una base de datos MySQL. La *shield* que extiende la funcionalidad básica de la placa permite, a su vez, la bidireccionalidad en la comunicación, necesaria para poder enviar peticiones desde el servidor a la API desarrollada en Arduino.

El servidor o *back-end* del proyecto, basado en Apache y desarrollado en PHP, contiene todo el conjunto de métodos que componen la API de la plataforma, que permiten leer/escribir en la base de datos o conectar con la placa Arduino en función de la petición enviada. Dicha API se presenta en un formato basado en REST, que posibilita que la interfaz de conexión sea fácilmente comprensible. Para ello se hace uso de un fichero PHP a modo de *router* que recibe y procesa las peticiones para encaminarlas al controlador relacionado. Las peticiones, enviadas en formato JSON, se redirigen a dicho fichero mediante el uso del módulo *rewrite* de Apache.

La parte de cliente o *front-end* está formada por una página creada con HTML5 y CSS3. Para facilitar la tarea de desarrollo se utiliza jQuery, que permite hacer llamadas AJAX de forma sencilla para recuperar o enviar datos al servidor, y a su vez introduce todo un conjunto de *plugins* que se utilizan en la implementación de la interfaz gráfica. Asimismo, se hace uso de la librería Bootstrap para desarrollar una página responsiva, así como de la fuente FontAwesome para incorporar iconos vectoriales en la aplicación.

Para permitir el envío de notificaciones en los dispositivos móviles se realiza un paquete de la página en PhoneGap de forma que el usuario pueda instalar la aplicación en su *smartphone*. Para el proyecto actual sólo se trabaja con Android, ya que ofrece mucha facilidad para instalar paquetes externos a Google Play mediante una simple dirección URL.

El proyecto introduce una complejidad adicional, y es que se debe proteger el canal de comunicación de forma que sólo pueda interactuar con el sistema Arduino el usuario identificado, ya que de lo contrario un usuario externo podría actuar sobre un sistema físico real y manipular sus parámetros o llevar a cabo procesos de forma malintencionada o errónea. Por ello, se desarrolla un sistema de *login* que permite autenticar al usuario, mostrándole una aplicación basada en su perfil o rol de acceso, y se protegen todas las llamadas a la API de forma que sólo se permitan cuando se envía un *token* válido de sesión.

Una vez finalizado el proyecto, se habrá creado una plataforma que será capaz de almacenar en una base de datos los valores enviados por un dispositivo electrónico (en este caso concreto, la piscina) para favorecer su consulta y su gestión.

En un futuro, la plataforma será más genérica para poder controlar múltiples dispositivos a la vez, generando así un servicio de *Internet of Things*. En una evolución posterior será necesario el registro como desarrollador de Apple de forma que la aplicación esté disponible también para dispositivos con sistema operativo iOS. Asimismo, resulta interesante la posibilidad de poder ofrecer esta aplicación en las tiendas oficiales. No obstante, este punto no adquiere gran relevancia en el trabajo actual al tratarse de un proyecto privado. De la misma manera, otra posible extensión consistiría en desarrollar la aplicación de forma nativa, utilizando los lenguajes de programación de cada sistema operativo.

## 3. Objetivos

### 3.1. Principales

Los objetivos principales que persigue este proyecto son:

- Recuperar y almacenar en una base de datos los valores recogidos por los sensores de un sistema Arduino.
- Diseñar una aplicación web que permita consultar la información almacenada en una base de datos.
- Actuar sobre un sistema Arduino de forma remota a través de Internet.
- Desarrollar una plataforma estable y segura.
- Reducir la complejidad de las tareas del usuario abstrayéndolo del funcionamiento real del sistema.

### 3.2. Secundarios

Además de los objetivos especificados en el punto anterior, se intentará dar respuesta a todos los objetivos secundarios que se detallan a continuación:

- Utilizar los conocimientos adquiridos en el grado para planificar y desarrollar un proyecto de gran envergadura.
- Diseñar una interfaz responsiva e intuitiva centrada en el usuario respondiendo a las bases del DCU.
- Conceptualizar e implementar una API clara e intuitiva.
- Administrar un servidor privado y generar certificados SSL para proteger las comunicaciones.
- Generar una memoria del proceso clara y consistente.
- Desarrollar una plataforma genérica que pueda expandirse posteriormente para su uso con más dispositivos, propia de una solución de *Internet of Things*.

## 4. Escenario

Nadie puede negar que Internet ha marcado un antes y un después en la forma en cómo se interrelacionan las personas. La tecnología evoluciona a un ritmo imparable y ya no sólo ha afectado a la vida de las personas, sino también a la simple existencia de los objetos, otorgándoles funcionalidades que les permiten conectarse entre sí y crear una enorme red de dispositivos que son capaces de comunicarse entre ellos y con nosotros.

Aunque este campo de la tecnología es relativamente “joven”, son muchos ya los dispositivos que existen en el mercado y a los que un usuario puede acceder sea cual sea su ubicación geográfica. Dispositivos inteligentes que permiten su gestión remota con sólo acceder a una página web o a una aplicación móvil. Acciones antes impensables como encender la lavadora de casa desde el trabajo están ahora a la orden del día. Y esto es sólo el principio, porque la era del Internet de las cosas no ha hecho más que empezar.

*Internet of things* abre un interesante abanico de posibilidades que necesita ser explotado y, de hecho, son ya varias las soluciones que permiten enviar la información a sus bases de datos para recibir información en tiempo real. Entre todas ellas destaca Xively, conocida antes como Cosm y todavía antes como Pachube, que lleva en el sector desde el 2007. En todos estos años ha conseguido registrar más de 255 millones de dispositivos en su base de datos, lo que refleja el enorme potencial que tienen los servicios de este estilo. Sin embargo, no son muchas las referencias que se pueden encontrar en las principales *stores* de dispositivos móviles, siendo éste un factor muy relevante hoy en día y que merecería la pena atacar.

Este proyecto pretende actuar a modo de trampolín para poder generar, en un futuro cercano, una plataforma que permita hablar con los dispositivos de igual forma que lo están haciendo servicios similares como Xively o ThingSpeak.

Es el momento, pues, de enrolarse en este nuevo y apasionante mundo con el fin de conectarse a estas tecnologías emergentes de la misma forma que ya lo están haciendo los objetos.

## 5. Contenidos

La plataforma funciona mayoritariamente con un tipo de contenido: los datos procedentes de los dispositivos que se conectan al sistema y que envían su información para almacenarla en la base de datos correspondiente.

A los datos se accede a través de la API del servidor, que devuelve la información solicitada por el usuario en formato JSON. Dicha información se trata en cliente y se organiza en bloques de gráficas que muestran la evolución de una medición a lo largo del tiempo. El intervalo temporal que muestra la gráfica es definible a petición del usuario, pudiendo así escoger el lapso de tiempo que se quiere mostrar en la representación lineal. El *dashboard*, que es la página inicial de la aplicación, muestra un resumen con el último valor de cada uno de los canales (pH u ORP, entre otros) y el estado del dispositivo.

Conceptualmente, la plataforma distingue los tipos de contenidos que se muestran a continuación:

- **User:** Un usuario de la plataforma.
- **Device:** Un dispositivo, un objeto que envía mediciones al sistema. En este caso, la piscina que se pretende gestionar.
- **Device methods:** Registra un método del dispositivo. Este método puede representar una acción sobre el sistema (*turnSystemOff* o *turnSystemAuto*, entre otros) o un parámetro configurable (*filteringStart* o *taRefCheck*, entre otros).
- **Channel:** Representa un canal compuesto por una colección de mediciones o *data points*. Por ejemplo, un canal podría ser el pH, el ORP o la temperatura del agua, que se componen de todas las mediciones que el dispositivo Arduino ha ido enviando al servidor de forma periódica.
- **Data point:** Un *data point* representa cada una de las mediciones que Arduino envía a la base de datos.



## 6. Metodología

El proyecto engloba muchas tareas que deben realizarse de forma secuencial –una detrás de otra– y paralela –varias al mismo tiempo–. Por ello, es de vital importancia definir un calendario realista con todos los hitos a tener en cuenta para poder valorar de la forma más objetiva posible si el proyecto avanza según lo previsto y aplicar medidas correctivas en caso contrario. Se empieza, por lo tanto, por definir el plan de trabajo mediante un diagrama de Gantt que muestra todas las fases con las actividades a llevar a cabo y su equivalencia temporal, haciendo encajar el modelo dentro de la planificación de entregas prevista por la asignatura.

Se realizarán revisiones periódicas para ver si el proyecto evoluciona de acuerdo con la planificación inicial, y se notificará al consultor cualquier incidencia que pueda haber en el *timing* para redefinir el plan de trabajo o para incrementar el número de entregas parciales.

Una vez analizadas las necesidades del usuario, se inicia un estudio de soluciones existentes de *Internet of Things*. En este caso, el estudio de servicios como *Xively* o *ThingSpeak* resultan de gran utilidad para diseñar y modelizar la API de la plataforma.

El período de evaluación se realizará con suficiente margen para poder corregir las incidencias detectadas en los juegos de pruebas. Teniendo en cuenta que el proyecto actúa sobre un entorno real, el consultor sólo dispondrá de una cuenta con acceso de lectura a las mediciones del sistema pero sin posibilidad de realizar acciones sobre los elementos físicos. Para paliar este inconveniente y poder evaluar la comunicación con la placa Arduino, se propone activar privilegios en esa cuenta de forma puntual y comprobar los resultados mediante una reunión síncrona a través de una videoconferencia, si fuese necesario.

Teniendo en cuenta que uno de los objetivos es facilitar las tareas del consumidor ofreciéndole una interfaz clara e intuitiva, será necesario tener siempre en mente que el diseño está centrado en el usuario. Se trabajará con *wireframes* de bajo nivel hasta dar con un diseño compositivo que responda a sus necesidades. No obstante, se priorizará la parte técnica de envío y recuperación de datos frente al desarrollo de la parte gráfica, que se dejará para la última fase del proyecto.

## 7.Arquitectura de la plataforma

### 7.1. Cliente

Esta capa utiliza:

- **jQuery**: Librería que permite desarrollar en JavaScript de forma mucho más eficiente y rápida.
- **HTML5**: Lenguaje de marcado para definir la estructura de la página.
- **CSS3**: Permite dar estilo a la página utilizando las últimas tendencias.
- **Bootstrap**: *Framework* que proporciona una forma fácil y flexible de componer un *layout* y de realizar un diseño responsivo, necesario para utilizar la página como una aplicación en los dispositivos móviles.
- **FontAwesome**: Permite incorporar fuentes iconográficas vectoriales de forma sencilla.
- **Navegador web**: Necesario para renderizar la página, independientemente del soporte. Merece la pena mencionar que Adobe PhoneGap está generando, al fin y al cabo, una aplicación con un *browser* que carga una página HTML. El hecho de disponer de un diseño responsivo es lo que permite presentar el contenido como si fuera una aplicación nativa.

### 7.2. Servidor

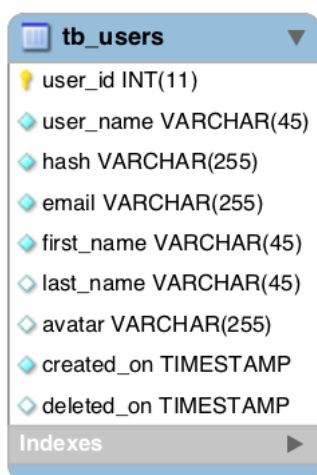
Está caracterizado por:

- **Servidor**: Máquina virtual Linux (tiene instalada la distribución de Ubuntu 12.04.2 LTS) con 1GB de almacenamiento, 2GB de memoria RAM y tráfico ilimitado, recursos suficientes como para poder ejecutar sin problemas las primeras versiones de la plataforma. Dispone de acceso FTP y de bases de datos ilimitadas.
- **Apache**: Servidor web de código abierto necesario para procesar las páginas y entregárselas al usuario.
- **PHP 5.3.10**: La API de servidor está programada con PHP, por lo que es necesario que el servidor web pueda tratar este lenguaje. Se requiere, a su vez, que tenga activado el soporte para la librería cURL, ya que es la utilidad que se emplea para contactar con Arduino a través de su REST API.

- **MySQL 5.5.31:** Soporte para bases de datos MySQL, necesario para almacenar toda la información de la plataforma.
- **Slim 2.4.2:** *Framework* para PHP que permite crear una API de forma ágil y sencilla.

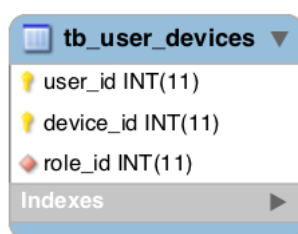
### 7.3. Base de datos

A continuación se muestran las tablas que componen la base de datos. El modelo entidad-relación no se incluye aquí ya que no resultaría legible debido a sus dimensiones, pero se entrega como anexo (Anexo1\_BDDModeloER.pdf).



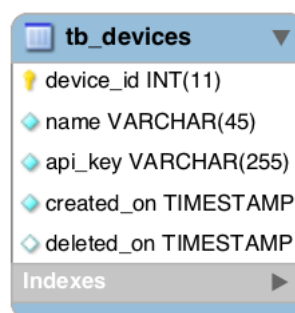
Field	Type
user_id	INT(11)
user_name	VARCHAR(45)
hash	VARCHAR(255)
email	VARCHAR(255)
first_name	VARCHAR(45)
last_name	VARCHAR(45)
avatar	VARCHAR(255)
created_on	TIMESTAMP
deleted_on	TIMESTAMP

Figura 2. Tabla de usuarios



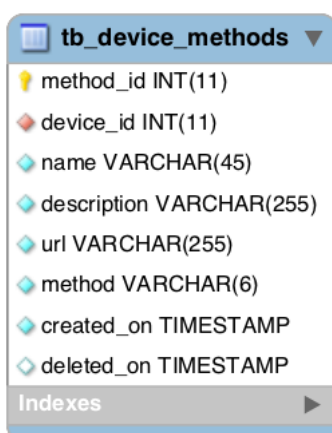
Field	Type
user_id	INT(11)
device_id	INT(11)
role_id	INT(11)

Figura 3. Tabla de dispositivos asociados al usuario



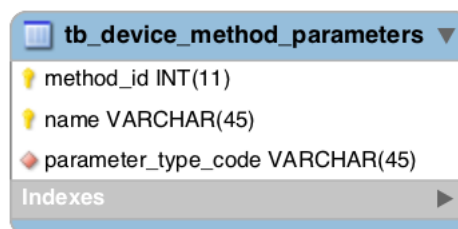
Field	Type
device_id	INT(11)
name	VARCHAR(45)
api_key	VARCHAR(255)
created_on	TIMESTAMP
deleted_on	TIMESTAMP

Figura 4. Tabla de dispositivos



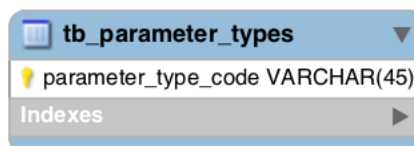
Field	Type
method_id	INT(11)
device_id	INT(11)
name	VARCHAR(45)
description	VARCHAR(255)
url	VARCHAR(255)
method	VARCHAR(6)
created_on	TIMESTAMP
deleted_on	TIMESTAMP

Figura 5. Tabla de métodos disponibles para el dispositivo



Field	Type
method_id	INT(11)
name	VARCHAR(45)
parameter_type_code	VARCHAR(45)

Figura 6. Tabla de parámetros para el método del dispositivo



Field	Type
parameter_type_code	VARCHAR(45)

Figura 7. Tabla de tipos de parámetros

tb_channels
channel_id INT(11)
device_id INT(11)
code VARCHAR(45)
name VARCHAR(45)
unit VARCHAR(10)
created_on TIMESTAMP
deleted_on TIMESTAMP
Indexes

Figura 8. Tabla de canales

tb_channel_labels
label_id INT(11)
channel_id INT(11)
value DOUBLE
label VARCHAR(45)
Indexes

Figura 9. Tabla de etiquetas para los valores de un canal

tb_datapoints
datapoint_id BIGINT(20)
channel_id INT(11)
value DOUBLE
date DATETIME
Indexes

Figura 10. Tabla de *data points* o mediciones

tb_channel_triggers
trigger_id INT(11)
user_id INT(11)
channel_id INT(11)
operator CHAR(1)
value DOUBLE
notif_mode VARCHAR(10)
created_on TIMESTAMP
deleted_on TIMESTAMP
Indexes

Figura 11. Tabla de disparadores a nivel de canal

tb_device_triggers
trigger_id INT(11)
device_id INT(11)
user_id INT(11)
max_inactivity BIGINT(20)
action CHAR(1)
created_on TIMESTAMP
deleted_on TIMESTAMP
Indexes

Figura 12. Tablas de disparadores a nivel de dispositivo

tb_mobile_devices
mobile_device_id INT(11)
user_id INT(11)
token VARCHAR(255)
platform VARCHAR(45)
version VARCHAR(45)
model VARCHAR(45)
uuid VARCHAR(45)
installed_on DATETIME
last_connection DATETIME
active INT(1)
Indexes

Figura 13. Tabla de dispositivos móviles

tb_notifications
notification_id INT(11)
user_id INT(11)
title VARCHAR(45)
message VARCHAR(255)
sent_on TIMESTAMP
handled_on TIMESTAMP
deleted_on TIMESTAMP
Indexes

Figura 14. Tabla de notificaciones enviadas

tb_roles
role_id INT(11)
name VARCHAR(45)
description VARCHAR(255)
Indexes

Figura 15. Tabla de roles de acceso

## 7.4. Arduino

Esta capa consta de:

- **Microcontrolador:** Arduino Mega 2560.
- **Sensores y actuadores:** Componentes electrónicos que permiten transmitir mediciones a la placa Arduino y que permiten controlar el estado de los elementos físicos del sistema como las bombas.
- **Webduino:** Librería que permite crear un servidor web para tratar las solicitudes especificadas por el usuario vía HTTP. Ha sido necesario introducir ciertos cambios en la librería para poder tratar peticiones en formato REST, que se especifican en el **Anexo 3. Librerías/código externo utilizado.**

## 8. Plataforma de desarrollo

### 8.1. Software

Durante el desarrollo de este proyecto se han utilizado los siguientes recursos de *software*:

- **Arduino IDE**: Entorno de desarrollo de Arduino. Necesario para poder codificar las clases, compilarlas y cargarlas en la placa.
- **PSPad** (Windows) y **TextWrangler** (MacOS): Editores de texto utilizados para llevar a cabo la codificación de la plataforma.
- **Cyberduck**: Gestor de servidores FTP con el que poder transferir archivos al servidor.
- **MySQL Workbench**: Herramienta para administrar la base de datos MySQL del servidor mediante una sencilla interfaz gráfica y de forma remota.
- **Adobe Illustrator**: Herramienta de dibujo vectorial para diseñar los elementos gráficos tanto de la interfaz como del presente documento.
- **Adobe PhoneGap**: Servicio para encapsular la página web del proyecto en un paquete que permita su instalación como aplicación en dispositivos móviles.
- **Microsoft Word 2011 para Mac**: Procesador de texto para realizar la memoria del proyecto.
- **Microsoft Project 2010**: Software de administración de proyectos utilizado para realizar el diagrama de Gantt.
- **Microsoft Remote Desktop**: Utilidad para acceder remotamente al portátil conectado al sistema Arduino.
- **Android Developer Tools**: Versión de Eclipse precompilada para funcionar correctamente con el SDK de Android.

### 8.2. Hardware

La maquinaria empleada ha sido:

- **iMac 27” – 64bits – 2.93 GHz Intel Core i7 – 8GB DDR3 – Mac OS X 10.6.8**: Ordenador de sobremesa desde el que se lleva a cabo todo el desarrollo y la redacción de la memoria.

- **Acer Aspire 5670 – 32 bits – 1,67GHz Intel Centrino Duo – 4GB RAM (3GB usables) – Windows 7 Ultimate:** Ordenador portátil que posee el entorno de desarrollo del sistema Arduino. Su uso es de vital importancia puesto que la placa está actualmente instalada en la sala de máquinas del domicilio y no es posible extraerla de su ubicación actual por largos periodos de tiempo. De no incorporar este ordenador sería necesario desarrollar *in situ*, y por motivos de tiempo, desplazamiento y comodidad resulta inviable.
- **Nexus 4 – 1,5GHz Qualcomm Snapdragon S4 Pro – 2GB RAM – Android 4.4.2 KitKat:** *Smartphone* con el que se realizan todas las pruebas de usuario de la aplicación móvil.
- **Arduino Mega 2560:** Placa Arduino a la que se conectan todos los sensores y actuadores del sistema. Microcontrolador basado en Atmeg1280. Cuenta con 54 entradas/salidas digitales (14 de ellas PWM), 16 entradas digitales, 4 puertos series y conexión USB.
- **Ethernet shield:** Permite extender las funcionalidades de Arduino ofreciéndole conectividad a Internet. Necesario para poder tratar las solicitudes remotas del usuario.
- **Servidor web Ubuntu 12.04.2 LTS:** Servidor virtual para alojar el *front-end*, el *back-end* y la base de datos. Dispone de 2GB de RAM garantizada, 100Mbps de conectividad, tráfico mensual ilimitado y 1GB de espacio de almacenamiento.

### 8.3. Otros servicios

También merece la pena destacar otros servicios que han resultado de utilidad para la consecución del proyecto:

- **Dropbox:** Servicio de almacenamiento *cloud* con el que organizar un repositorio virtual de todo el contenido del proyecto de forma que sea posible mantener una copia de seguridad.
- **Creately:** Herramienta *online* de creación de diagramas, utilizado para plasmar de forma gráfica la representación de las diferentes API de la plataforma.
- **Moqups:** Herramienta *online* de creación de *wireframes* de baja fidelidad.
- **Parallels Panel 11.5.30:** Servicio *online* para administrar el servidor privado.

## 9. Planificación

La planificación prevista incluye todas las fases del desarrollo del proyecto, desde el análisis de necesidades hasta su puesta en marcha definitiva. Arranca de la mano del inicio del semestre, el **26 de febrero del 2014**, y se finaliza el **24 de junio del 2014**.

### 9.1. Fechas clave

A lo largo del proyecto se plantean tres fechas inamovibles que corresponden a las entregas parciales de la memoria y del material que hasta el momento se haya generado y que, por lo tanto, hay que tener en cuenta en la planificación temporal.

<i>Fecha</i>	<i>Concepto</i>	<i>Entregables destacados</i>
11/03/2014	PAC1	Primera versión de la memoria
06/04/2014	PAC2	Segunda versión de la memoria Prototipo de acceso a Arduino vía HTTP Diseños UML
11/05/2014	PAC3	Tercera versión de la memoria API de la plataforma Propuesta gráfica de la plataforma Primeras interacciones con el servidor desde cliente
24/06/2014	Entrega final	Versión final de la memoria Versión final de la plataforma Presentaciones e informes

**Tabla 1.** Fechas clave del proyecto

### 9.2. Hitos

Además de las cuatro entregas clave que coinciden con las PACs de la asignatura se establecen también otros hitos parciales que determinan el progreso satisfactorio del proyecto.



Fecha	Concepto
19/04/2014	Desarrollo del <i>back-end</i> finalizado
14/05/2014	Desarrollo del <i>front-end</i> finalizado
24/05/2014	Juego de pruebas finalizado
31/05/2014	Corrección de errores finalizada
21/06/2014	Proyecto finalizado

Tabla 2. Hitos parciales del proyecto

### 9.3. Diagrama de Gantt

La figura que se incluye a continuación muestra el calendario relacionado con este proyecto. Se incluye también como recurso adjunto para ofrecer una versión legible (Anexo1\_DiagramaGantt.pdf).

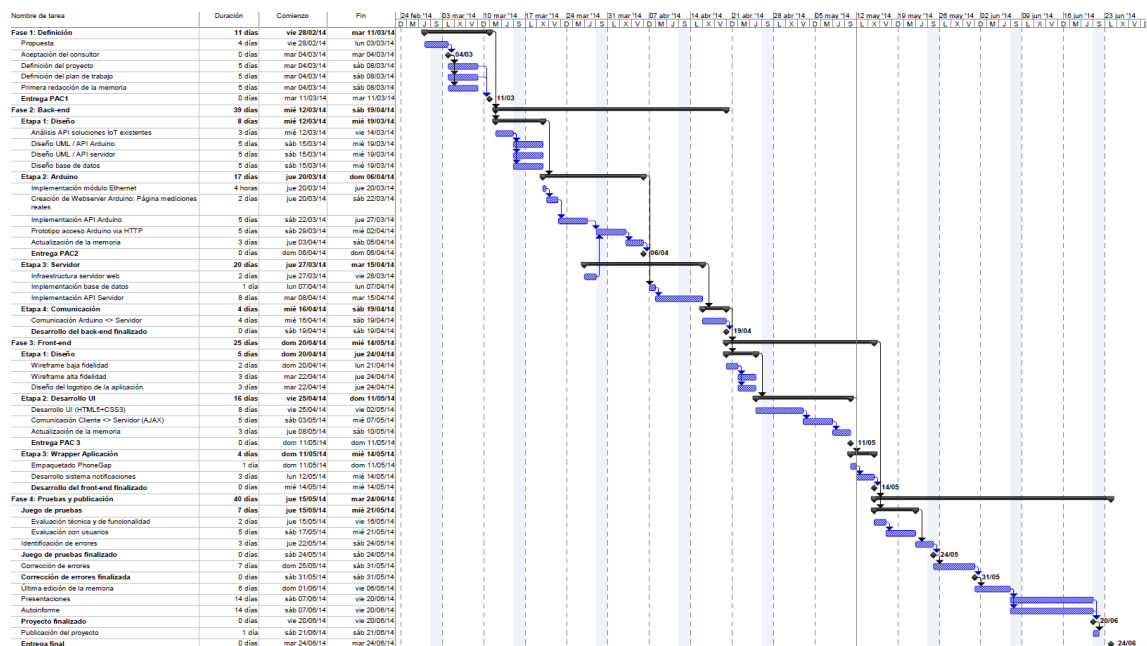


Figura 16. Diagrama de Gantt del proyecto

## 10. Proceso de desarrollo

El proceso de desarrollo seguido para la consecución de este proyecto es el siguiente:

### 10.1. Diseño inicial

La primera fase empieza con un breve análisis para ver cuáles son las mejoras prácticas en servicios similares al que propone este proyecto. Tras detectar cuáles son los patrones comunes, se empieza a diseñar la estructura de clases de Arduino y del servidor y se modeliza la base de datos. Teniendo en cuenta la complejidad en la lectura del código actual de Arduino, desarrollado con programación estructural, se decide diseñar un conjunto de clases para tener una organización lógica e intuitiva de los módulos físicos del sistema.

### 10.2. Conexión con Arduino

El dispositivo Arduino del sistema inicial no posee conectividad a Internet. La infraestructura de la que se parte sólo lee mediciones de los sensores cuando el usuario se lo solicita y las muestra en una pantalla LCD conectada a la placa. La opción de gestión automática permite que las bombas se pongan en marcha a una hora específica y durante un tiempo definido directamente dentro del código de la plataforma. Esto supone un gran inconveniente cuando se requiere cambiar el valor de dicho parámetro, ya que en ese caso se necesita abrir el código fuente del programa para localizar y modificar la variable y volver a cargar el *sketch* en la placa Arduino. A su vez, el emplazamiento actual dificulta mucho el desarrollo, ya que el dispositivo se encuentra en una sala de máquinas que ofrece un entorno poco agradable para trabajar. Tampoco es posible desconectar la placa temporalmente ya que cada modificación implicaría volver a conectar todos los elementos físicos a sus pines. Además, es necesario que la placa esté conectada a todos los sensores para codificar, ya que de lo contrario no se puede saber con exactitud si el código funciona tal y como se esperaba.

Viendo todas estas limitaciones, el primer paso consiste en dejar un ordenador portátil conectado al microcontrolador mediante un cable USB. Al habilitar la conexión por escritorio remoto se puede acceder al entorno de desarrollo desde unas mejores condiciones. Hecho esto, la siguiente tarea consiste en añadir una placa Ethernet al Arduino que permita

conectarlo a la red local. Tras asignarle una dirección IP, el dispositivo se añade como un equipo de la red.

En este punto se carga la librería *Webduino*, que permite crear un servidor web, y se define una primera versión de acceso al dispositivo que devuelve un texto con las mediciones en tiempo real. Sin embargo, sólo es posible acceder a esta página a través de la red local, y es necesario poder contactar con Arduino desde cualquier lugar. Por ello se requiere abrir un puerto en el *router* que apunte a la dirección IP interna y al puerto definido para el servidor web, en este caso el 80.

Dado que la librería *WebDuino* no está pensada para tratar peticiones en formato REST, es necesario hacer una pequeña modificación en el método *dispatchCommand*, que trata la solicitud del cliente para interpretar correctamente la ruta especificada. El código modificado se describe en el apartado **Anexo 3: Librerías/código externo utilizado**.

En este punto se empiezan ya a codificar las clases diseñadas en la fase anterior. Al acabar, el servidor tramita las peticiones HTTP solicitadas por el usuario y devuelve la respuesta en el formato JSON deseado.

Durante el proceso, el módulo RTC externo del sistema Arduino deja de funcionar súbitamente. Aprovechando la nueva conectividad a Internet, se desarrolla un sistema adicional de sincronización mediante el protocolo NTP, de forma que la placa tenga la fecha bien informada en todo momento. Esta fecha es de vital importancia para arrancar los procesos en el momento indicado y, más adelante, para enviar los datos al servidor sabiendo exactamente cuándo se tomaron.

Acabada la integración del módulo Ethernet, el siguiente paso consiste en preparar el servidor creando una primera versión de la infraestructura necesaria para tratar las solicitudes realizadas desde el prototipo de comunicación con el dispositivo Arduino. Se utiliza para ello un servidor virtual del que ya se dispone con anterioridad.

Se genera una primera versión de la base de datos, que por ahora contiene una única tabla de usuarios con la que almacenar información sobre las credenciales de acceso a la

plataforma. En lugar de guardar la contraseña en texto plano se guarda un *hash* obtenido mediante la función de encriptación *encrypt*, lo que hace su gestión más segura.

En un fichero de configuración de PHP se guardan los datos de conexión a esa base de datos y la dirección IP pública del Arduino, entre otros.

Se crean entonces dos páginas HTML: una con la estructura del *login* y otra con la estructura necesaria para poder definir una solicitud al Arduino. Para no tener que memorizar cuáles son los puntos de entrada de la REST API se genera un *array* con todas las rutas, que se cargan posteriormente como opciones en el desplegable relacionado.

En PHP se prepara un archivo que contiene las funciones de acceso al dispositivo, así como el *login* y el *logout*. Se genera también un fichero *index.php* a modo de *router*, que comprueba si el usuario está identificado o no y le redirige a la pantalla de *login* en caso negativo. En función de las credenciales del usuario se bloquea el acceso a los métodos PUT, POST y DELETE –aunque éste último no se utilice en ningún caso–. En cuanto al *login*, la comprobación de la validez de los datos se efectúa recuperando el valor del atributo *hash* asociado al usuario que intenta iniciar sesión y comparándolo con uno nuevo calculado a partir de la contraseña especificada en el formulario de acceso.

Mediante una llamada AJAX desde el cliente se accede al método deseado del servidor. En caso de contactar satisfactoriamente con el dispositivo Arduino, se formatea la respuesta para mejorar su legibilidad.

La dirección del prototipo acabado y los datos de conexión se especifican a continuación:

<http://sergileon.com/arduino/>

Usuario **kenneth**

Contraseña **tfg%UOC**

## 10.3. Implementación del servidor y de la base de datos

Partiendo de la definición del modelo Entidad-Relación de la base de datos y del UML del servidor, se empieza el desarrollo de ambos entornos.

Este punto se aborda en dos fases separadas: la primera consiste en la implementación del esquema de la base de datos y las primeras inserciones para poder probar posteriormente la REST API a generar; la segunda corresponde a la implementación del modelo de la API diseñado en la primera etapa del proyecto.

La creación de la estructura de tablas y relaciones de la base de datos resulta muy sencilla al haber utilizado MySQL Workbench para generar el modelo Entidad-Relación, ya que sólo es necesario ejecutar la traducción del modelo hacia la base de datos de destino. Una vez creada la estructura, el siguiente paso consiste en rellenar la información de las tablas principales.

Para la implementación de la REST API del servidor se parte de un *framework* para PHP conocido como Slim, que permite definir la interfaz de entrada de la API de una forma rápida y sencilla.

Establecidos los puntos de entrada, se codifican los diferentes métodos de la *PoolDuinoAPI*. Se crean las clases que permiten obtener información de cada entidad, como por ejemplo la clase *Devices* o la clase *Channels*. Todos los métodos se crean de forma estática para no tener que instanciar ningún objeto previamente, pudiendo acceder directamente a las funciones que se conectan a la base de datos y recuperan la información solicitada.

Para verificar que sólo los usuarios autorizados puedan recuperar información de la base de datos se implementa una clase de comprobación de cabeceras, que permite comprobar que se reciben los *tokens* necesarios para autenticar al usuario o dispositivo que accede a la API.

Cualquier error controlado se lanza con la clase *PoolDuinoAPIException*, en la que se definen los posibles errores que se gestionan en la aplicación de forma que no sea necesario escribir el mensaje y el código cada vez que se codifica una excepción.

En esta fase se codifica también una clase *Database* que permite gestionar una conexión con una base de datos MySQL, capaz de lanzar excepciones personalizadas de tipo *DatabaseException* ante errores inesperados.

Por último se desarrolla también una clase *Log* que permite dejar trazas en el servidor para poder analizar la ejecución de los métodos.

## 10.4. Comunicación entre Arduino y el servidor

En este punto del proyecto ya se ha creado la comunicación de servidor hacia Arduino, pero ésta todavía no es bidireccional. Es decir, en el prototipo previo ya se lograba acceder a la API creada en Arduino para modificar los parámetros de configuración, para realizar acciones o para obtener mediciones en tiempo real. Ahora es necesario crear un método adicional en la API del dispositivo con el que enviar datos al servidor cada cierto tiempo. Para ello se crea la función *sendData*, que obtiene la medición actual de los diferentes canales definidos en el sistema y los encapsula en formato JSON para enviarlos al servidor a través del método POST habilitado para tal efecto. La clase API mantiene también el valor de la clave *ApiKey*, que necesita enviar en la cabecera de cada petición para que el servidor autorice la inserción de datos desde el dispositivo.

## 10.5. Diseño de la interfaz y el logotipo

El primer paso de esta fase consiste en saber qué se quiere mostrar en cada pantalla, seleccionando la información que se desea transmitir y agrupándola en categorías intuitivas para el usuario. Tan pronto como se conocen estos detalles se empieza con el proceso de conceptualización de la aplicación, dibujando primero las diferentes pantallas en papel para acabar dando con un diseño compositivo que encaje con los requerimientos a los que se pretende dar respuesta.

El siguiente paso consiste en traspasar estos diseños a mano a *wireframes* de baja fidelidad, con los que acabar de ajustar la composición básica. Para ello se hace uso de la herramienta *online* Moqups. Acto seguido se transforman los *wireframes* obtenidos aplicando el diseño gráfico final con la combinación de colores y tipografías, haciendo uso del software de edición gráfica vectorial Adobe Illustrator.

El logotipo sigue el mismo esquema de trabajo. Se conceptualizan varias ideas en papel para posteriormente pasar a su versión digital vectorial mediante la misma herramienta gráfica del punto anterior.

El resultado de esta fase se puede ver con más detalle en el apartado [13.2. Wireframes de baja fidelidad](#), [13.3 Wireframes de alta fidelidad](#) y [13.4 Logotipo](#).

## 10.6. Desarrollo de la interfaz de usuario

Partiendo de los *wireframes* de alta fidelidad elaborados en la tarea anterior, el siguiente paso consiste en desarrollar la versión para navegador asociada al diseño generado.

Para abordar esta fase se divide el trabajo en dos partes: la primera subtarea consiste en la elaboración del código HTML y CSS estático de cada pantalla; la segunda consiste en aplicarle interactividad solicitando datos al servidor mediante llamadas AJAX con jQuery y tratando la respuesta para generar los bloques de código que se han obtenido en la primera parte.

Se empieza, pues, a diseñar la estructura básica de la aplicación. En este punto se incluyen todos los *plugins* que permiten generar los componentes complejos de la interfaz como las gráficas, las ventanas modales o los selectores de fecha y hora, para poder otorgarles el estilo que encaja con el del *wireframe* de alta fidelidad. También se crea la navegación básica entre pantallas, aplicando los efectos de desplazamiento lateral y vertical que se invocan al pulsar sobre las opciones de menú.

Con la estructura creada, el siguiente paso consiste en solicitar datos dinámicos al servidor mediante llamadas AJAX a la REST API creada. El código HTML estático generado en la fase anterior se construye ahora desde las funciones JavaScript para poder crear elementos en función de la respuesta del servidor. Con esto se consigue ofrecer contenidos dinámicos que permiten monitorizar y gestionar el sistema Arduino gracias a la visión de canales y a la pantalla de acciones.

La dirección de la aplicación y los datos de conexión se especifican a continuación:

<http://poolduino.sergileon.com/>

Usuario **kenneth**

Contraseña **tfg%UOC**

Merece la pena mencionar que, en una versión definitiva, un usuario de tipo consultor no podrá ver las acciones disponibles para el dispositivo. No obstante, para poder valorar la evolución del proyecto se le permite tener acceso a dicha sección y lo único que se restringe es la ejecución final.

## 10.7. Desarrollo del sistema de notificaciones

El primer paso en esta etapa consiste en darse de alta como desarrollador en Google para poder tener acceso a la API de *Google Cloud Messaging* con la que habilitar el envío de notificaciones *push* hacia el dispositivo del usuario. Tras finalizar el registro, se crea un nuevo proyecto, habilitando el módulo GCM, y se obtienen las claves necesarias para el envío y recepción de notificaciones.

A continuación se genera el paquete PhoneGap para su instalación en el dispositivo móvil. Se descarga y configura el entorno de desarrollo utilizando Android Development Tools, una versión precompilada de Eclipse que facilita la integración con la SDK de Android. Se configura posteriormente la extensión de PhoneGap, así como el *plugin* GCM que permite el tratamiento de notificaciones y su acceso desde JavaScript.

El funcionamiento de la aplicación es sencillo: al abrirse se genera un navegador que carga el contenido del fichero local *index.html*, que a su vez posee una redirección hacia la ruta <http://poolduino.sergileon.com/index.php?mode=app>. El parámetro *mode* recibido por *QueryString* indica si la petición procede del terminal móvil, cargando así las librerías relacionadas que permiten el acceso a la información del dispositivo. De esta manera, la misma página se reutiliza tanto para su acceso desde dispositivos móviles como desde navegadores convencionales. Esto facilita la gestión, ya que no es necesario replicar los cambios en dos entornos ante cualquier modificación en el código.



Se modifica posteriormente el procedimiento de acceso a la plataforma, ya que ahora no sólo debe realizar el *login* sino también el registro del dispositivo de forma que la base de datos pueda almacenar el *token* del dispositivo con el que habilitar el envío de notificaciones *push* hacia el terminal. El cierre de sesión también se modifica, ya que debe desvincular el dispositivo del usuario para no seguir enviándole notificaciones.

Acto seguido, se detallan las diferentes reglas ante las cuales es necesario notificar al usuario, creando los registros relacionados en la tabla *tb\_channel\_triggers*. Se realiza el control pertinente ante la inserción de un nuevo *datapoint*, comprobando si su valor requiere la creación de un aviso. Asimismo, se genera la función que permite notificar al usuario, ya sea con una notificación *push* o mediante un correo electrónico, en función de cómo se haya definido el método de aviso en la base de datos.

Hecho todo esto, se carga el paquete APK al servidor y se crea una sencilla página HTML que actúa como *landing page* con la que facilitar la descarga de la aplicación en el terminal del usuario. Dicha página es accesible a través de <http://poolduino.sergileon.com/download>. Una vez descargada e instalada, será posible enviarle notificaciones ante los eventos que haya definido:

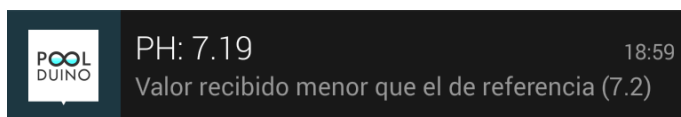


Figura 17. Ejemplo de notificación

## 10.8. Detección y corrección de errores

En esta fase se comprueba el correcto funcionamiento de la aplicación en los soportes para los que se ha desarrollado. Se llevan a cabo los juegos de prueba y se analizan todas las incidencias detectadas, que se detallan en el apartado **22. Incidencias**. Tras llevar a cabo las acciones correctivas pertinentes, se obtiene la primera versión definitiva del aplicativo, siendo ahora completamente funcional.

## 11. APIs utilizadas

Para el desarrollo del proyecto se han codificado tanto la API en formato REST de Arduino como del servidor, que se detallan en los siguientes apartados.

### 11.1. API Arduino

Esta API permite interactuar con el dispositivo Arduino para solicitar las mediciones actuales, para cambiar el modo de funcionamiento o para configurar sus parámetros de uso. Los puntos de entrada que se han definido se describen a continuación:

<i>Método</i>	<i>Endpoint</i>	<i>Descripción</i>
GET	/api/status	Devuelve un resumen con la situación actual de todos los canales.
GET	/api/status/mode	Devuelve el valor del modo de funcionamiento.
GET	/api/status/ta	Devuelve el valor de la temperatura del agua.
GET	/api/status/te	Devuelve el valor de la temperatura exterior.
GET	/api/status/ph	Devuelve el valor del pH.
GET	/api/status/orp	Devuelve el valor del ORP.
GET	/api/status/excesssolar	Devuelve el estado del exceso solar.
GET	/api/status/filteringpump	Devuelve el estado de la bomba de filtración.
GET	/api/status/chlorinepump	Devuelve el estado de la bomba de dosificación de cloro.
GET	/api/status/taref	Devuelve el valor de la temperatura de referencia.
GET	/api/info	Devuelve la hora del dispositivo y su versión de <i>firmware</i> .
PUT POST	/api/actions/doseChlorine	Dosifica cloro durante los segundos especificados por parámetro.
PUT POST	/api/actions/turnSystemOff	Cambia el modo del sistema a OFF.
PUT POST	/api/actions/turnSystemAuto	Cambia el modo del sistema a AUTO.
PUT POST	/api/actions/turnSystemManual	Cambia el modo del sistema a MANUAL.

GET	/api/config	Devuelve un resumen de todos los parámetros de configuración.
GET	/api/config/webpost	Devuelve el tiempo que pasa entre cada envío de datos.
PUT POST	/api/config/webpost	Cambia el tiempo que pasa entre cada envío de datos.
GET	/api/config/tarefcheck	Devuelve la hora a la que se comprueba la temperatura de referencia.
PUT POST	/api/config/tarefcheck	Cambia la hora a la que se comprueba la temperatura de referencia.
GET	/api/config/filteringstart	Devuelve la hora a la que se pone en marcha automáticamente la bomba de filtración.
PUT POST	/api/config/filteringstart	Cambia la hora a la que se pone en marcha automáticamente la bomba de filtración.

**Tabla 3.** Puntos de entrada de la API de Arduino

## 11.2. API Servidor

Esta API contiene la lógica de la aplicación, incluyendo tanto los métodos de autenticación de usuarios y dispositivos como las funciones de acceso e inserción de datos procedentes del sistema Arduino. También contiene las rutinas que permiten interactuar con el dispositivo para realizar acciones en el sistema físico. Los puntos de entrada se describen a continuación:

<i>Método</i>	<i>Endpoint</i>	<i>Descripción</i>
GET	/api/devices	Devuelve la lista de dispositivos.
GET	/api/devices/id	Devuelve un resumen del dispositivo.
POST	/api/devices/id	Recibe una colección de <i>channel-datapoints</i> para su inserción en la base de datos. Sólo disponible desde el dispositivo.
GET	/api/devices/id/channels	Devuelve la lista de canales del dispositivo.
GET	/api/devices/id/channels/id	Devuelve un resumen del canal con sus <i>datapoints</i> . Es posible limitar la respuesta mediante el parámetro <i>from</i> y <i>to</i> .
GET	/api/devices/id/channels/id/datapoints	Devuelve la lista de <i>datapoints</i> del canal. Es posible limitar la respuesta mediante el parámetro <i>from</i> y <i>to</i> .
GET	/api/devices/id/methods	Devuelve la lista de métodos del dispositivo.

GET	/api/devices/id/methods/id	Devuelve un resumen del método.
PUT	/api/devices/id/methods/id/run	Ejecuta el método en base a los parámetros necesarios para su ejecución.
GET	/api/user/info	Devuelve información del usuario conectado.
GET	/api/user/notifications	Devuelve la lista de notificaciones del usuario.
GET	/api/user/notifications/id	Devuelve información sobre la notificación.
PUT	/api/user/notifications/id/handle	Trata la notificación para marcarla como leída, no leída o borrada en función del parámetro <i>action</i> .
POST	/api/auth/request_token	Solicita un nuevo <i>token</i> e inicia sesión.
POST	/api/auth/invalidate_token	Finaliza la sesión asociada al <i>token</i> .
POST	/api/auth/register_device	Registra un nuevo dispositivo móvil para el envío de notificaciones <i>push</i> .
POST	/api/auth/unregister_device	Desactiva el dispositivo especificado.

**Tabla 4.** Puntos de entrada de la API de Servidor

### 11.3. SDK Android

Para poder generar un paquete instalable en dispositivos Android es necesario utilizar la SDK que Google pone al alcance de los desarrolladores. De esta manera se puede codificar una aplicación nativa y generar el paquete para instalarlo en terminales Android. Se emplea esta SDK en la creación de la estructura de ficheros, así como en la configuración de permisos del paquete, la gestión de notificaciones, la ejecución de una máquina virtual para probar los avances en el ordenador y la compilación final de la aplicación.

### 11.4. API PhoneGap

Esta API encapsula todas las funciones que permiten interactuar fácilmente con el dispositivo a través de JavaScript. En este proyecto se hace un uso bastante básico de este paquete, ya que sólo lo utilizamos para conocer información básica del terminal como el modelo y la versión de Android que tiene instalada, entre otros, así como para adquirir el *token* que necesitamos para poder enviar notificaciones *push* al dispositivo. Relacionado con este punto se incluye el *plugin* oficial GCM de PhoneGap, que añade todo el control que debe realizar el terminal de forma nativa para obtener las notificaciones del servidor de Google y mostrar la alerta pertinente en la barra de tareas del dispositivo móvil.

## 12. Diagramas UML

### 12.1. Arduino

El diagrama UML que refleja las clases del sistema Arduino, así como sus relaciones, se puede consultar en el archivo adjunto correspondiente ([Anexo1\\_ArduinoUML.pdf](#)). No se incorpora en este apartado debido a las limitaciones de resolución que supone incrustarlo en este documento, ya que el volumen de información es demasiado elevado y no resulta legible.

### 12.2. Servidor

Tal y como sucede en el apartado anterior, el volumen de información de este diagrama es demasiado grande, afectando directamente a la legibilidad del mismo. Se puede consultar en el archivo adjunto correspondiente ([Anexo1\\_ServidorUML.pdf](#)).

## 13. Prototipos

### 13.1. Conexión con Arduino

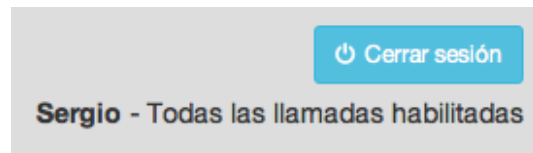
En la primera fase del proyecto se crea un primer prototipo de comunicación con Arduino para comprobar que todas las comunicaciones se pueden ejecutar de forma correcta mediante la REST API. El primer prototipo contempla ya el módulo de acceso o *login* que autentica al usuario y le restringe o le permite ciertas solicitudes en función de sus privilegios.



El prototipo de la pantalla de login de Arduino muestra un encabezado con el título "Arduino". Debajo, se encuentra un formulario con el título "Conéctate". El formulario incluye un campo de entrada para el "Usuario", un campo de entrada para la "Contraseña" y un checkbox con el texto "Recordar mis datos". En la parte inferior del formulario hay un botón azul con el texto "Entrar" y un ícono de checkmark.

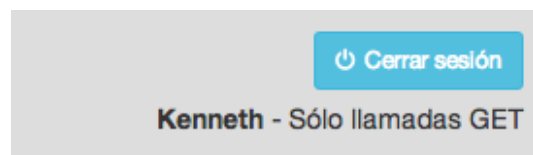
Figura 18. Prototipo: Pantalla de *login*

Tras acceder, el usuario puede realizar las solicitudes de lectura o modificación seleccionando el *end point* deseado, el método a ejecutar y escribiendo los parámetros que espera recibir el dispositivo, si es que son necesarios para la acción escogida.



El prototipo de la pantalla de usuario administrador muestra un botón azul con el texto "Cerrar sesión" y un ícono de power. Debajo del botón, se muestra el nombre del usuario "Sergio" y su nivel de privilegios "Todas las llamadas habilitadas".

Figura 19. Prototipo: Usuario administrador



El prototipo de la pantalla de usuario consultor muestra un botón azul con el texto "Cerrar sesión" y un ícono de power. Debajo del botón, se muestra el nombre del usuario "Kenneth" y su nivel de privilegios "Sólo llamadas GET".

Figura 20. Prototipo: Usuario consultor

Este prototipo muestra un formulario para configurar una solicitud. Incluye un campo 'End point' con el valor 'api/status/mode', un campo 'Método' con el valor 'GET', y un campo 'Parámetros' con el ejemplo '(P.e.: param1=value1&param2=value2)'. Un botón azul 'Enviar' está situado en la parte inferior.

Figura 21. Prototipo: Datos de la solicitud

Este prototipo muestra una respuesta JSON en un recuadro gris. El código es: 

```
{  "name": "mode",  "value": "auto"}
```

Figura 22. Prototipo: Respuesta de la solicitud

## 13.2. Wireframes de baja fidelidad

En la segunda fase del proyecto se crea un primer esbozo de la información que se mostrará en la aplicación y cómo se organizará el contenido en cada pantalla. La realización de los *wireframes* de baja fidelidad se lleva a cabo mediante el servicio gratuito Moqups accesible desde <https://moqups.com>. Se puede acceder a una versión de mayor resolución en el anexo Anexo1\_WireframesBajaFidelidad.pdf.

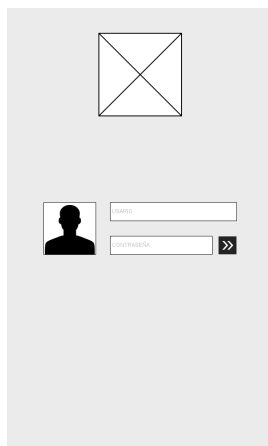


Figura 23. Wireframe baja fidelidad: Login

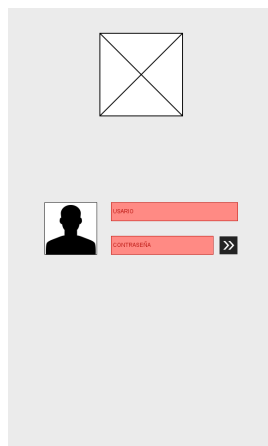


Figura 24. Wireframe baja fidelidad: Login – Campos vacíos

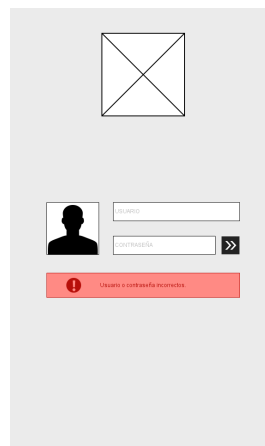


Figura 25. Wireframe baja fidelidad: Login – Error de credenciales



Figura 26. Wireframe baja fidelidad: Bloqueo de sesión



Figura 27. Wireframe baja fidelidad: Dashboard

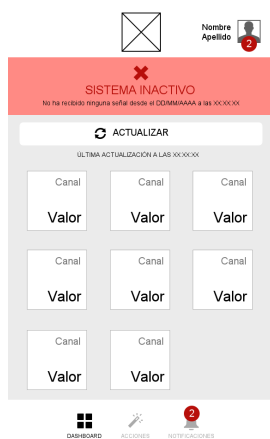


Figura 28. Wireframe baja fidelidad: Dashboard – Sistema caído



Figura 29. Wireframe baja fidelidad: Canal



Figura 30. Wireframe baja fidelidad: Canal – Selección de intervalo

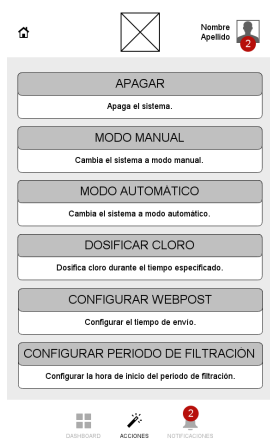


Figura 31. Wireframe baja fidelidad: Acciones



Figura 32. Wireframe baja fidelidad: Acciones – Ejecución

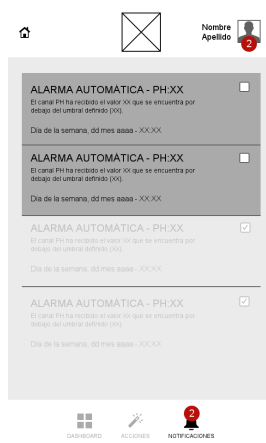


Figura 33. Wireframe baja fidelidad: Notificaciones

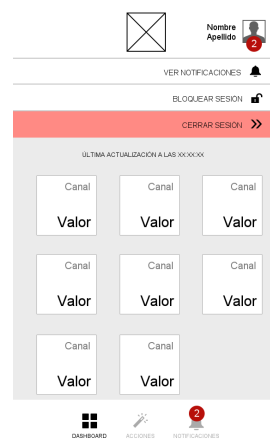


Figura 34. Wireframe baja fidelidad: Acciones de sesión

### 13.3. Wireframes de alta fidelidad

Una vez diseñada la estructura compositiva de la aplicación se define la interfaz gráfica definitiva, implementando todos los estilos, tipografías y colores a utilizar. Para ello se utiliza el programa Adobe Illustrator, que permite manipular gráficos vectoriales fácilmente adaptables a cambios de resolución. Se realiza un diseño minimalista basado en el uso de colores planos. Se puede acceder a una versión de mayor resolución en el anexo Anexo1\_WireframesAltaFidelidad.pdf.



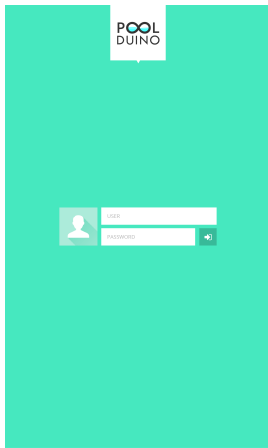


Figura 35. Wireframe alta fidelidad: Login

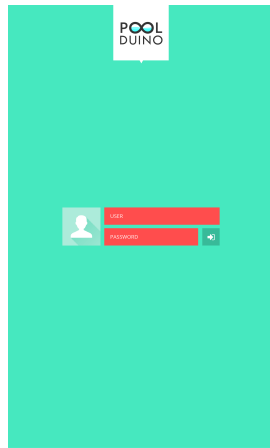


Figura 36. Wireframe alta fidelidad: Login – Campos vacíos

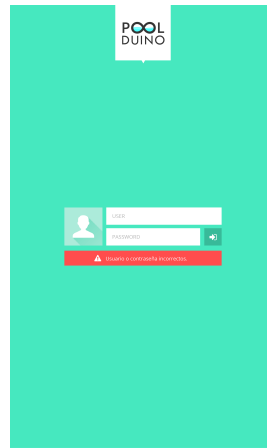


Figura 37. Wireframe alta fidelidad: Login – Error de credenciales

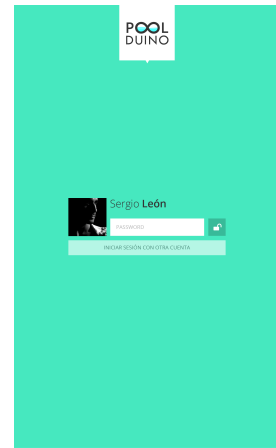


Figura 38. Wireframe alta fidelidad: Bloqueo de sesión

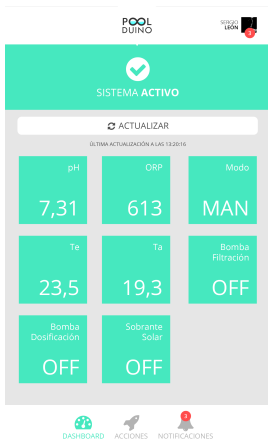


Figura 39. Wireframe alta fidelidad: Dashboard



Figura 40. Wireframe alta fidelidad: Dashboard – Sistema caído



Figura 41. Wireframe alta fidelidad: Canal

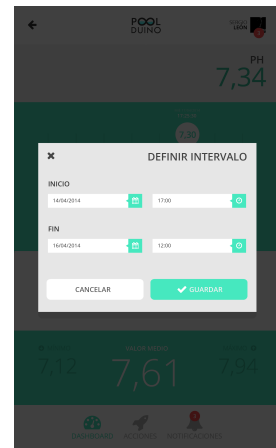


Figura 42. Wireframe alta fidelidad: Canal – Selección de intervalo



Figura 43. Wireframe alta fidelidad: Acciones



Figura 44. Wireframe alta fidelidad: Acciones – Ejecución

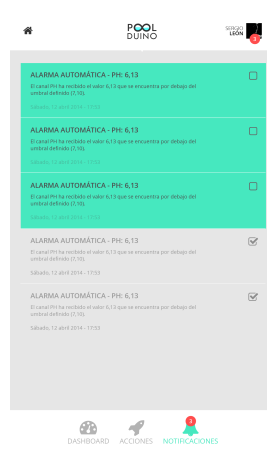


Figura 45. Wireframe alta fidelidad: Notificaciones

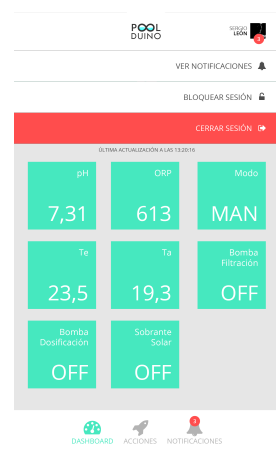


Figura 46. Wireframe alta fidelidad: Acciones de sesión

## 13.4. Logotipo

En la segunda fase del proyecto se diseña también el logotipo de la aplicación, a la que se bautiza como PoolDuino. El diseño se nutre de ambos términos, ya que las dos *o* que forman la palabra *pool* están inspiradas en el logotipo de Arduino, que a su vez define un contenedor lleno de agua relacionada con la piscina.



Figura 47. Logotipo

## 13.5. Página de descarga

Tal y como ya se ha comentado en el apartado **10. Proceso de desarrollo**, para facilitar la descarga de la aplicación en los terminales Android se diseña una sencilla página HTML:

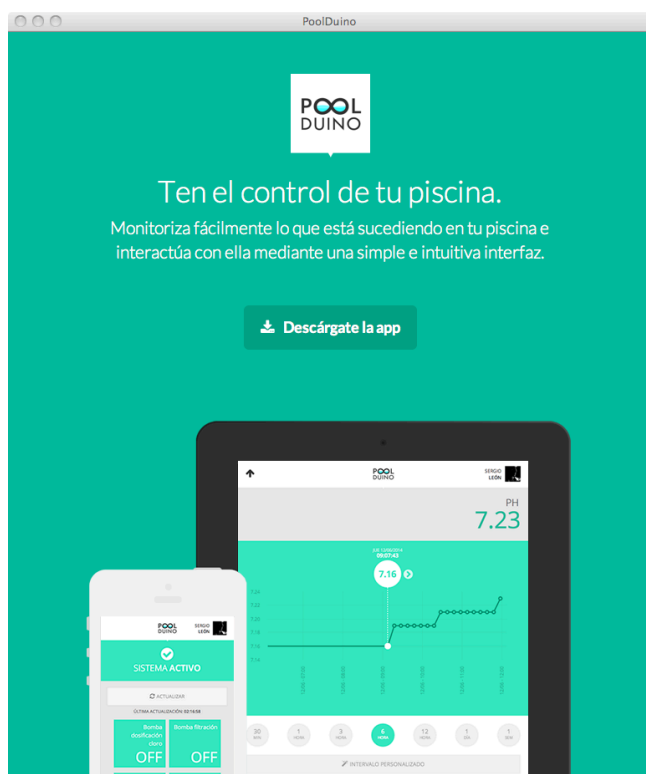


Figura 48. Página de descarga

## 14. Perfiles de usuario

Este proyecto plantea una solución muy concreta y por lo tanto se enfoca a un rango de usuarios muy reducido. En este caso la solución se ha diseñado para gestionar un dispositivo propio, y por lo tanto responde mayoritariamente a las necesidades del propietario. No obstante, el diseño de la plataforma se ha desarrollado teniendo en cuenta su posible extensión como servicio de *Internet of Things*, y como tal, atacará a un público objetivo de las siguientes características:

- **Usuarios individuales** que disponen de un dispositivo que permite enviar datos a la red y desean monitorizarlo. Dentro de este grupo encontramos usuarios que han adquirido el dispositivo o desarrolladores *amateurs*.
- **Empresas desarrolladoras y fabricantes de *software* y *hardware*** que implementan nuevos dispositivos conectados a Internet.

Estos dos puntos repercuten en un grupo formado mayoritariamente por personas de entre 18 a 50 años con intereses muy concretos. Se trata de un sector que siente la necesidad de adaptarse a las nuevas tecnologías. En cuanto al género, la plataforma no se enfoca a ningún sexo en particular.

La aplicación no posee ningún interés para alguien que no encaja con este *target*, ya que todos los datos que recopila la plataforma se muestran de forma privada a los usuarios que tienen acceso a dichas mediciones. Es decir, no existe información pública –al menos de momento–, por lo que un usuario nuevo no vería nada en su panel.

Los usuarios que acceden a este servicio se caracterizan por los privilegios que poseen sobre el dispositivo, existiendo dos grandes roles:

- **Usuario administrador:** Este tipo de usuario posee un control total sobre los objetos de la plataforma, pudiendo recuperar todos los datos y ejecutar acciones que tenga definidas el dispositivo.
- **Usuario consultor:** Este tipo de usuario sólo puede recuperar datos pero no puede realizar ninguna acción ni modificación sobre el sistema.

## 15. Usabilidad

El diseño de la interfaz de la plataforma se realiza teniendo en cuenta las bases del DCU, por lo que se modeliza respetando las necesidades del usuario. Hay que tener en cuenta que el acceso más importante a la plataforma será a través de dispositivos móviles, así que merece la pena no perder de vista las limitaciones que eso supone.

*Grosso modo*, a continuación se exponen los valores que se han intentado respetar:

- La interfaz se diseña con la intención de que resulte muy **intuitiva** y fácil de utilizar, **minimizando la curva de aprendizaje** al máximo.
- Los botones se diseñan respetando un **hit area suficientemente grande** como para ser fácilmente seleccionables con los dedos.
- Se utiliza un **diseño plano**, con **colores visualmente agradables** y respetando la **coherencia** entre pantallas.
- Las **fuentes** son **fácilmente legibles** con tamaños suficientemente grandes para no forzar la vista.
- Todas las gráficas del *dashboard* repiten la **misma estructura**, que otorga coherencia al diseño y hace sentir cómodo al usuario porque sabe cómo llega la información que espera leer.
- Se utiliza un **patrón compositivo** muy bien asimilado por el usuario: **cabecera** con el logotipo de la aplicación y botones de acción básicos (menú y configuración); una **zona de contenido** por la que se puede desplazar mediante *scroll* vertical; y **pie estático** con opciones de navegación.
- Siempre que el usuario realiza una acción se **notifica el progreso mediante un mensaje o spinner** en pantalla, que le hace saber que su petición se está tratando y el sistema no se ha quedado bloqueado. Una vez finalizada, se le notifica el estado de la solicitud de forma que pueda saber si ha funcionado o, por el contrario, se ha producido algún error.
- Se utiliza una **jerarquía muy reducida** ya que de lo contrario se desorientaría al usuario y se le haría perder información contextual.
- Se utiliza un **etiquetado claro** con el que usuario pueda asimilar rápidamente qué función desempeña cada apartado.

## 16. Seguridad

La plataforma se ha diseñado teniendo en cuenta la importancia de la protección de los datos de acceso al sistema Arduino. No hay que olvidar que se trata de un dispositivo que es capaz de manipular un entorno físico real, por lo que resulta de vital importancia poder restringir el acceso a usuarios que no disponen de los privilegios necesarios para accionar sobre el sistema ya que podrían hacer un uso malintencionado.

En primer lugar, pues, se desarrolla el módulo de *login* que permite autenticar al usuario, prohibiendo así el acceso anónimo a la plataforma. Hecho esto, se desarrolla un sistema de roles que permite asignarle unos privilegios concretos. Con ello se le conceden permisos parciales o totales sobre el sistema, permitiéndole acceso de sólo lectura (obtener información sobre mediciones pasadas o en tiempo real) o de modificación (realizar acciones de forma remota, modificar parámetros de configuración, etc). El primer caso es el de la cuenta proporcionada al consultor, que tan sólo tiene habilitadas aquellas funcionalidades que no modifican el estado del sistema Arduino.

Para ofrecer más seguridad, la base de datos no almacena contraseñas legibles a simple vista sino que guarda un *hash* generado mediante técnicas de encriptación, de forma que no sea posible extraer la palabra de paso original del usuario. El acceso a la plataforma compara el *hash* guardado en la base de datos con el obtenido mediante la encriptación de la contraseña suministrada al método *login*. En caso de coincidir, esta función le devuelve un *token* que deberá incluir en las cabeceras de todas las llamadas posteriores a la API, de forma que se realicen las acciones sólo si el usuario está correctamente autenticado.

Relacionado con el punto anterior encontramos un mecanismo de seguridad en cuanto a la inserción de mediciones en el sistema. Para verificar la integridad de los *datapoints* que se insertan en la base de datos se asigna una clave *ApiKey* al sistema Arduino. Cada llamada a la API del servidor envía una cabecera con esta clave de modo que permita autenticarle.

Seguidamente, se incluyen validaciones en cada una de las capas (*front-end*, *back-end* y Arduino) de forma que sea posible validar la integridad de los valores introducidos por el usuario y que se envían a las peticiones que actúan sobre Arduino. Es el caso de

modificaciones sobre parámetros de configuración o acciones que requieran algún parámetro adicional (como por ejemplo la dosificación de cloro, que requiere el número de segundos que debe estar en marcha la bomba relacionada). Estas validaciones permiten evitar posibles valores incorrectos que podrían hacer que el sistema funcionara de forma indeseada.

Por otro lado, al ser la API del servidor la que gestiona la conexión con Arduino, es posible ocultar la IP del dispositivo al usuario final, abstrayéndolo así de los datos que realmente se envían y de la URL a la que se dirigen.

Uno de los objetivos secundarios del proyecto, aunque fundamental en cuestiones de seguridad, era poder establecer una conexión segura con el dispositivo Arduino mediante un certificado SSL que permitiera cifrar los datos de las solicitudes que éste debe gestionar. Sin embargo, la placa actual (Mega 2560) no dispone de un procesador con la potencia necesaria como para poder asumir la carga computacional que supone la encriptación y desencriptación de la información. Se ha encontrado, no obstante, una interesante solución alternativa: el intercambio del microcontrolador por una placa Arduino YUN, que dispone de un módulo Linux que sí que proporciona la capacidad suficiente como para establecer una conexión segura. No obstante, esta modificación escapa del alcance inicial y se valora como un aspecto de proyección a futuro.

También relacionado con este apartado se encuentran las copias de seguridad del progreso del proyecto. Se opta por utilizar Dropbox para almacenar *backups* de cada versión de la plataforma y de la memoria, de forma que no se pierdan datos de forma irrecuperable.

## 17. Juegos de pruebas

Con la finalidad de poder valorar la eficiencia de la plataforma se proponen los siguientes juegos de pruebas.

### 17.1. Evaluación técnica

Permite validar que la aplicación funciona correctamente en todos los soportes para la que ha sido programada, en concreto para dispositivos móviles y para ordenadores. Ésta debe funcionar en la mayoría de navegadores web actuales como Chrome, Firefox o Safari, y adaptar correctamente su diseño para resoluciones limitadas en su visualización desde *smartphone*. Esta fase se encarga también de validar los aspectos de seguridad en la comunicación y de estabilidad de la plataforma. Con estos tests se detectan pequeñas incompatibilidades entre navegadores, que se detallan en el apartado **22. Incidencias**.

### 17.2. Evaluación de funcionalidad

Su objetivo es comprobar que la plataforma desarrolla todos los requerimientos especificados en el proyecto y que éstos funcionan de la forma esperada. Los análisis llevados a cabo confirman que todas las funcionalidades detalladas están desarrolladas satisfactoriamente.

### 17.3. Evaluación con usuarios

Se selecciona un grupo de personas que encajan con el perfil de usuario de la plataforma para que realicen pruebas sobre el sistema con el fin de analizar su experiencia de usuario y comprobar la eficacia del diseño de usabilidad. En este punto, se les solicita realizar acciones específicas y se guarda un registro de todos los aspectos en los que hayan encontrado alguna dificultad. Ninguno de los usuarios tiene ningún problema en acceder a la información de un canal, filtrar por intervalos concretos o realizar acciones sobre el sistema, entre otros. Sin embargo, en algunos casos se detecta que la parte de notificaciones no queda completamente clara visualmente en relación a los colores utilizados, así que se reestructura el apartado gráfico y se añade una nueva funcionalidad a petición de los usuarios: la posibilidad de eliminar notificaciones para borrarlas de la lista.

## 18. Versiones de la plataforma

### 18.1. Versión Alpha

Corresponde a la primera etapa de conexión con el sistema Arduino. Contiene:

- **Reestructuración código Arduino:** Desarrollo del código actual en clases bien diferenciadas y fácilmente comprensibles.
- **API Arduino:** Librería para gestionar las instancias de todos los elementos físicos del sistema desde un único punto de entrada.
- **REST API Arduino:** Interfaz de comunicación REST con el dispositivo. Interactúa con la clase API de Arduino.
- **Primera infraestructura del servidor:** Codificación de la primera capa del servidor para tratar las solicitudes entrantes (*login*, *logout* y *requests* de Arduino). Gestiona *cookies* para mantener la sesión abierta.
- **Primera versión de la base de datos:** Necesaria para almacenar los datos de usuarios y sus roles.
- **Módulo de login:** Módulo necesario para autenticar al usuario.
- **Prototipo de comunicación:** Página simple para realizar una solicitud REST al dispositivo. Permite especificar el *end point*, el método de la solicitud y los parámetros, si fueran necesarios. Formatea la respuesta JSON devuelta por el dispositivo para hacer más sencilla su lectura.

### 18.2. Versión Beta

Corresponde a la segunda etapa del proyecto, que consiste en el desarrollo del *back-end* y del *front-end*. Contiene:

- **Implementación de la base de datos:** Implementación del modelo entidad-relación definido en la etapa de diseño.
- **Implementación API del servidor:** Desarrollo de las clases de *back-end* para insertar la información procedente del dispositivo Arduino y para obtener los datos por parte de la interfaz de cliente.



- **Envío de datos procedentes de Arduino:** Desarrollo de clase para realizar las solicitudes de inserción de datos en el repositorio MySQL accediendo a la API del servidor.
- **Estructura del *front-end*:** Desarrollo de la capa de conexión con el servidor y de la interfaz gráfica de la página.

### 18.3. Versión 1.0 - Release

Corresponde a la última etapa del proyecto. Contiene:

- **Desarrollo de control de eventos:** Módulo que permite detectar cuando se ha producido un suceso que requiere una notificación al usuario.
- **Desarrollo de sistema de notificaciones:** Módulo de envío de notificaciones *push* a los dispositivos móviles.
- **Detección y corrección de errores**
- **Versión definitiva de la aplicación:** Paquete APK para instalación en dispositivo Android.

## 19. Requisitos de instalación

### 19.1. Cliente desde navegador

Al tratarse de una aplicación diseñada para una página web, es suficiente con tener:

- **Navegador compatible con HTML5 y JavaScript activado:** La mayoría de navegadores actuales disponen de compatibilidad con HTML5 y traen el módulo JavaScript activado por defecto.
- **Conexión a Internet**

### 19.2. Cliente desde dispositivo móvil

En caso de querer instalar la aplicación en el dispositivo será necesario disponer de:

- **Dispositivo Android con versión mínima 3.1 (API Level 12)**
- **Conexión a Internet**

Para todos aquellos dispositivos que no satisfacen los requisitos mencionados, tales como terminales iOS o versiones de Android incompatibles, es posible acceder a la plataforma directamente desde el navegador del dispositivo. Para ello se aplicarán los mismos requisitos que los especificados en el apartado anterior, **19.1. Cliente desde navegador**.

### 19.3. Servidor

Al tratarse de una aplicación web se requiere:

- **Servidor web Apache con módulo PHP instalado:** La aplicación contiene páginas con código PHP y HTML, así que requiere compatibilidad con dichos lenguajes.
- **Servidor de base de datos MySQL:** Necesario para almacenar los datos del dispositivo de forma relacional.
- **Procesador, almacenamiento, memoria y ancho de banda:** Es importante disponer de recursos suficientes para ofrecer un servicio estable y rápido.

### 19.4. Arduino

Este proyecto sólo aborda la parte de *software* del sistema Arduino, así que sólo requiere:

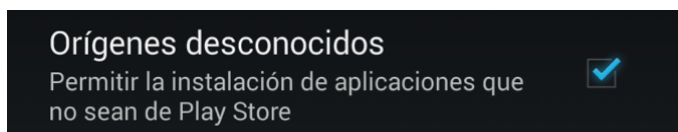
- **Arduino IDE:** Entorno de desarrollo con el que codificar e implementar el *sketch* en la placa Arduino.

## 20. Instrucciones de instalación

En este apartado sólo se detallan las instrucciones de instalación del paquete en dispositivos móviles Android, ya que para el uso de la aplicación desde navegador sólo es necesario disponer de un *browser* compatible instalado.

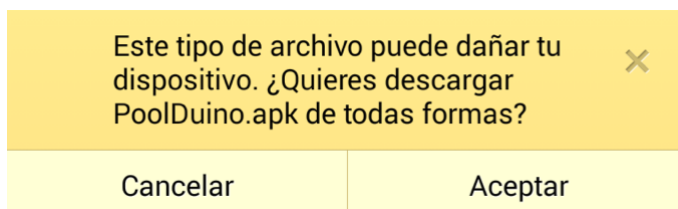
Los pasos que hay que seguir para la instalación de la aplicación en terminales móviles compatibles son:

- 1. Habilitar la opción Orígenes desconocidos:** Es necesario permitir la instalación de paquetes con fuentes desconocidas, ya que la aplicación no se ha publicado en la Play Store. Para ello se accede a Ajustes > Seguridad y marcamos la casilla *Orígenes desconocidos*.



**Figura 49.** Instalación: Casilla de orígenes desconocidos

- 2. Descargar el paquete de PoolDuino:** Se debe acceder a la página web <http://poolduino.sergileon.com/download> y pulsar sobre el botón *Descárgate la app*. Aparecerá un mensaje indicando que el archivo podría ser dañino para el sistema al no descargarse de una fuente segura.



**Figura 50.** Instalación: Confirmación de descarga

Al pulsar *Aceptar* se iniciará la descarga del paquete en el dispositivo.

3. **Instalar el paquete:** Una vez descargado es necesario ejecutar el archivo desde la notificación recibida o desde el apartado de *Descargas* del dispositivo. Aparecerá un resumen de los permisos que solicita el paquete.

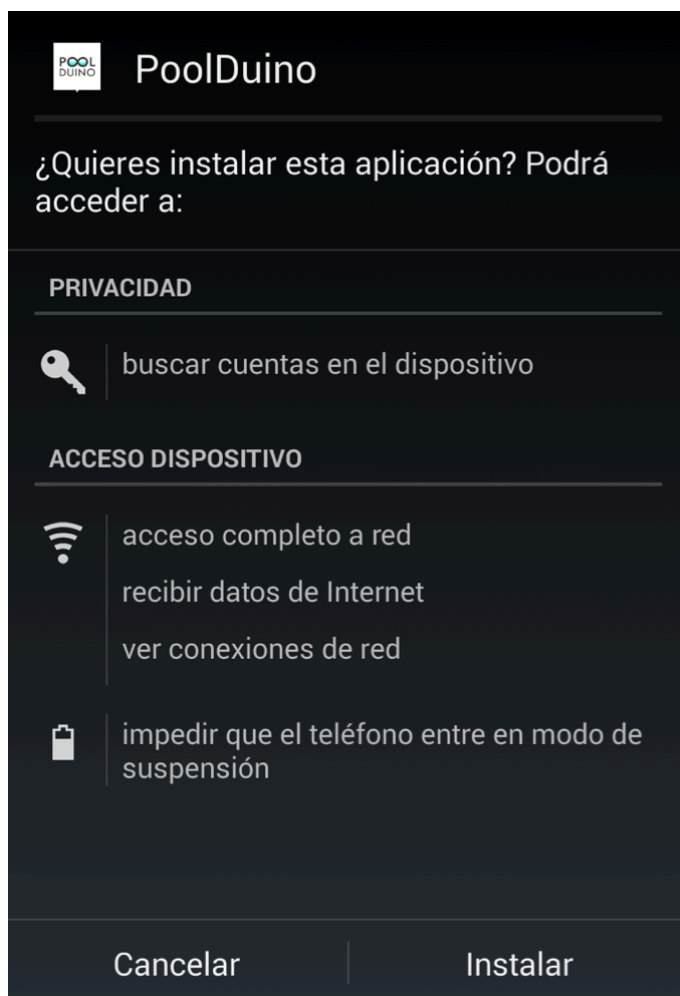


Figura 51. Instalación: Permisos

Al pulsar *Instalar* y terminar el proceso ya se dispondrá de la aplicación instalada en el dispositivo.

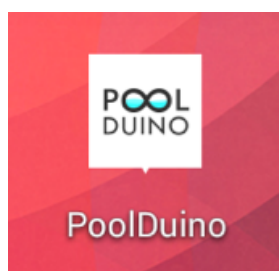


Figura 52. Instalación: Icono de la aplicación

## 21. Instrucciones de uso

Esta sección se muestra con más detalle en el [Anexo 5. Guía de usuario](#). Aquí se realiza, sin embargo, un pequeño resumen de los pasos principales para utilizar la plataforma:

- Asegurarse de que el dispositivo (ordenador o terminal móvil) dispone de conectividad a Internet.
- Abrir el navegador y acceder a <http://poolduino.sergileon.com> o ejecutar la aplicación en caso de que se haya instalado el paquete en el dispositivo móvil.
- En la pantalla de *login*, introducir las credenciales de acceso y pulsar *Enter* o el botón de conexión del formulario.
- Una vez dentro de la plataforma se habilitan todas las secciones de la aplicación:
  - a. Dashboard:** Muestra una visión de resumen del estado actual del dispositivo y de todos sus canales. Al presionar sobre estos bloques se accede a información específica del canal en cuestión. El usuario puede saber a simple vista en qué situación se encuentra el sistema.
  - b. Canal:** Muestra información detallada sobre el canal seleccionado. Incluye una gráfica de evolución de las mediciones, un bloque de selección de intervalo e información de resumen sobre los valores del periodo. El usuario puede conocer, de esta manera, detalles concretos del canal para un momento determinado.
  - c. Acciones:** Muestra las diferentes acciones que se pueden ejecutar sobre la plataforma. Al pulsar sobre el botón de cada una se abre una ventana que permite acceder a información adicional de la acción e introducir cualquier parámetro necesario. De esta forma, el usuario puede interactuar fácilmente con la piscina, abstrayéndole del mecanismo de comunicación. La ejecución está restringida a aquellos usuarios que dispongan de permisos suficientes.
  - d. Notificaciones:** Muestra las alertas que se han enviado al usuario sobre las reglas que ha introducido en el sistema, y se le permite tratarlas o eliminarlas.
  - e. Usuario:** Muestra información del usuario conectado y las opciones para finalizar su sesión. También incluye un acceso directo hacia el apartado de notificaciones.

## 22. Incidencias

Durante la fase de pruebas se han detectado pequeñas incidencias fruto de haber desarrollado toda la aplicación centrándose en un único navegador para acelerar el proceso. Así, al intentar consultar los datos de un canal mediante Firefox o Safari, la página notifica de la existencia de un *script* que está tomando más tiempo del esperado:

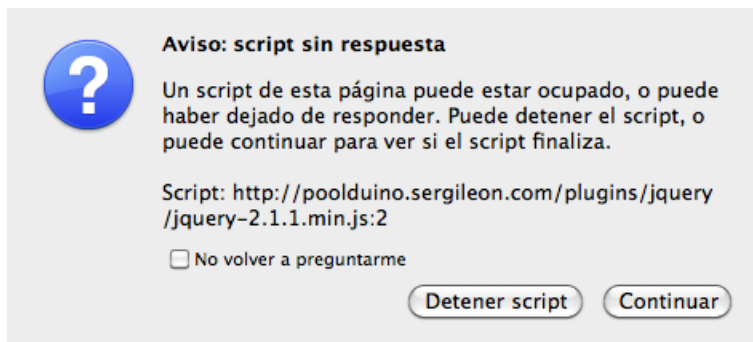


Figura 53. Incidencias: *Script* sin respuesta

Tras analizar la fuente del error se detecta que el problema reside en la función recursiva *adjustFont*, que permite ajustar dinámicamente el tamaño de la fuente para que el texto quepa sin problemas dentro de su contenedor. Con el código inicial, los navegadores mencionados no son capaces de recuperar el nuevo tamaño de la etiqueta HTML que contiene el texto, y por lo tanto se genera un bucle infinito. Se modifica el código de dicha función para solucionar esta casuística, tal y como se muestra a continuación:

```
1595 function adjustFont(elem) {
1596     var fontSize = elem.css('font-size').substr(0,2);
1597     var maxHeight = elem.height();
1598     var maxWidth = elem.width();
1599     var textHeight;
1600     var textWidth;
1601     do {
1602         elem.css('font-size', fontSize);
1603         textHeight = elem.height();
1604         textWidth = elem.width();
1605         fontSize = fontSize - 0.2;
1606     } while (textHeight > maxHeight || textWidth > maxWidth);
1607 }
```

Código 1. Función *adjustFont* para ajuste automático de tamaño de fuente – app.js

Por otro lado, se detecta que en la aplicación para Android no funciona correctamente el sistema de autenticación basado en *cookies*. El paquete generado con PhoneGap no es capaz de actualizar el contenido de la *cookie* una vez creada y por lo tanto nunca refresca el valor del *token* de sesión, necesario para autorizar el acceso a la plataforma. Para solventar este inconveniente se opta por utilizar almacenamiento de datos local mediante el objeto *localStorage* de HTML5, con el que se guardan todos los datos de sesión del usuario sin problemas tanto en la versión para *smartphone* como para la de escritorio.

Pese a que técnicamente no es una incidencia producida a raíz del desarrollo, merece la pena mencionar que el módulo RTC (*Real Time Clock*) conectado al sistema Arduino deja de funcionar justo en el inicio de la primera fase del proyecto. Éste era el encargado de gestionar la fecha y hora de la placa, necesarias para el correcto funcionamiento del sistema. Aprovechando la inclusión de la *Ethernet shield*, se soluciona este inconveniente desarrollando una clase de sincronización vía NTP (*Network Time Protocol*) que permite obtener la información horaria de un servidor web. Dado que el reloj interno de Arduino no es muy exacto, se realiza una resincronización cada hora para tener siempre el valor más preciso posible.

## 23. Proyección a futuro

Como ya se ha comentado en el apartado **16. Seguridad**, uno de los objetivos del proyecto era proporcionar un entorno de comunicación seguro entre el servidor y el dispositivo Arduino. No obstante, la placa actual no tiene la capacidad de procesamiento necesaria como para poder soportar conexiones encriptadas mediante un certificado SSL. Es por ello que uno de los aspectos a mirar en un futuro sería cambiar la placa Mega 2560 por una YUN, que dispone de un módulo Linux capaz de establecer conexiones mediante este protocolo de seguridad. Además, el hecho de tener dos unidades de procesamiento separadas permitiría poder liberar al Arduino de la lógica del servidor web, centrándose sólo en la gestión del entorno físico. No obstante, son varias las limitaciones que han impedido poder incorporar este módulo en el sistema de forma inminente:

- Por un lado, YUN dispone de **un único puerto serial de entrada**, cuando en estos momentos la infraestructura actual requiere tener al menos dos para poder conectar las sondas de pH y ORP. La única solución a este problema sería emplear un multiplexador especial que permitiera conectar varios componentes electrónicos a un mismo puerto.
- La compra de los dos módulos necesarios, adquiridos en distribuidores separados, representa un importante desembolso económico e introduce una nueva complejidad en el proyecto: los **tiempos de envío**. Una demora en la entrega podría suponer un importante desvío en la planificación del proyecto, ya que sin disponer de los dos componentes no se podría iniciar el desarrollo de la primera fase.

Otro aspecto muy relevante para el correcto funcionamiento de la plataforma en caso de mantener un número muy elevado de transacciones simultáneas sería poder hospedar el servicio en un **servidor cloud escalable**. De esta forma sería posible adaptar los recursos en función de las necesidades del cliente, aumentando la capacidad de almacenamiento o la RAM del servidor, entre otros.

Un punto adicional que sería muy interesante añadir en el futuro es la creación de una **plataforma de gestión** que permitiera dar de alta nuevos dispositivos, administrando intuitivamente su configuración y sus *triggers* disponibles, los usuarios del sistema y sus privilegios de acceso. Actualmente, toda esa información se gestiona directamente a través



de la base de datos ya que la creación de dicha plataforma supondría un proyecto nuevo en cuanto a volumen de trabajo. No obstante, la evolución lógica y deseada de este trabajo es presentarlo como un servicio web de *Internet of Things*, así que resulta absolutamente indispensable desarrollar esta extensión.

Teniendo en cuenta que ahora mismo la aplicación para dispositivos móviles se trata de un paquete que sólo actúa como *wrapper* de una página web remota, una extensión muy interesante sería **desarrollar la aplicación en el lenguaje de programación propio de cada sistema operativo**, de forma que ésta sea realmente nativa. Sería relevante estudiar si algún *framework* como Appcelerator Titanium ofreciera las funcionalidades necesarias como para codificar en un solo lenguaje y que éste hiciera la compilación hacia cada uno de los lenguajes específicos, simplificando el proceso de desarrollo.

También sería necesario **registrarse como desarrollador de Apple** de forma que fuera posible generar el paquete para la *App Store*, con el pago de las licencias que esto conlleva. Este proyecto sólo ha explotado la posibilidad que ofrece Android para poder instalar paquetes de fuentes desconocidas, lo que hace imposible su instalación en iOS.

Otro punto a mejorar que debería llevarse a cabo en la aplicación, especialmente importante en caso de expandir su uso, es incorporar la posibilidad de **escoger el idioma** en el que se presenta la información, ya que todos los textos se han codificado directamente en castellano.

Por último, sería interesante añadir herramientas para realizar **copias de seguridad** de la información almacenada en MySQL, permitiendo así poder recuperar el mayor volumen posible de datos en caso de una pérdida accidental. Teniendo en cuenta que el sistema operativo es Linux, su desarrollo sería relativamente sencillo utilizando herramientas como *cron* y *rsync*, que permiten la automatización de tareas y la sincronización de archivos entre carpetas.

## 24. Presupuesto

<i>Tarea</i>	<i>Perfil</i>	<i>Horas</i>	<i>Coste*</i>
<b>Análisis</b>			
Definición del alcance	Jefe de proyecto	40	1.120,00
Definición del plan de trabajo			
<b>Diseño</b>			
Base de datos	Jefe de proyecto	64	1.792,00
API Arduino			
API Servidor			
Wireframes	Diseñador gráfico	40	720,00
Logotipo			
<b>Implementación</b>			
Infraestructura técnica	Jefe de proyecto	14	392,00
Base de datos	Programador	8	144,00
API Arduino	Programador	96	1.728,00
API Servidor	Programador	80	1.440,00
Desarrollo UI	Diseñador gráfico	40	720,00
Empaquetado PhoneGap	Jefe de proyecto	4	112,00
Sistema notificaciones	Programador	24	432,00
<b>Puesta en marcha</b>			
Juego de pruebas y correcciones	Jefe de proyecto	80	1.920,00
Corrección de errores	Diseñador gráfico		
<b>Documentación</b>			
Memoria, informes y guías	Jefe de proyecto	88	2.464,00
Presentación audiovisual	Diseñador gráfico	24	432,00
<b>Gastos adicionales</b>			
Servidor (contratación anual)			179,40
<b>Presupuesto final</b>			<b>13.595,40</b>

\* Todos los precios especificados en EUR

## 25. Conclusiones

Una vez concluido el proyecto es el momento de realizar un análisis retrospectivo de todo el trabajo llevado a cabo.

Para empezar, este proyecto me ha resultado apasionante no sólo por el interés personal en la idea de la que partía, sino también por los múltiples soportes con los que he tenido que trabajar, cada uno con sus ventajas e inconvenientes. Si bien es cierto que en cada fase se han dado problemas que han requerido mi completa atención, ha sido muy interesante poder encontrar la forma de irlos solventando para acabar generando la plataforma a la que podemos acceder hoy en día.

La recodificación del sistema Arduino del que se partía y la creación de las APIs de *back-end* ha sido una parte muy interesante que, aunque ha requerido muchas horas de dedicación, he disfrutado gratamente. Construir la forma en la que comunicarnos con ambos entornos ha sido una ardua tarea, pero ver cómo se ponía en marcha un mecanismo físico con sólo lanzar un comando ha resultado algo asombroso que, desde luego, ha merecido la pena.

El diseño y el desarrollo de la interfaz de cliente ha sido una de las partes que más he disfrutado, ya que aúna dos de mis pasiones. Son muchos los aspectos a mejorar, eso es innegable, pero creo que he conseguido generar una plataforma visualmente agradable e intuitiva, que facilita enormemente el interés principal del proyecto: gestionar la piscina de forma sencilla.

La fase de empaquetado y envío de notificaciones ha resultado ser bastante más compleja de lo que pensaba, ya que el proyecto resulta algo “atípico” al no incluir el código en la aplicación. Esto ha requerido un esfuerzo por dar con un sistema en el que, incluyendo librerías en servidor, se pudiera acceder a la información del dispositivo y recodificar los procedimientos de inicio y cierre de sesión. Relacionado con este punto me he encontrado con que la navegación por la aplicación móvil no es tan fluida como en navegadores de escritorio, lo que me hace pensar que sería muy interesante llevar a cabo el desarrollo de forma nativa con tal de mejorar la usabilidad y calidad del producto final. Teniendo en

cuenta que uno de los objetivos era ofrecer una interfaz compatible con dispositivos móviles y ordenadores de sobremesa, la realización de este punto era inviable debido al escaso tiempo con el que se contaba. Sin embargo, sería sin duda uno de los puntos más importantes del sistema en cuanto a un desarrollo posterior.

A lo largo del proyecto creo haber podido aplicar muchos de los conocimientos que he podido adquirir durante el grado, tanto en temas de diseño gráfico como de programación y seguridad. En lo referente a la documentación creo que he llevado a cabo un trabajo minucioso que ha sido posible gracias a la segmentación en las entregas, que sin duda ha ayudado a repartir la carga durante el transcurso del proyecto.

Ciertamente, es mucho el esfuerzo que ha solicitado el proyecto, aunque ha sido muy gratificante poder concebirlo desde cero y comprobar el resultado final mientras escribo estas líneas. Desarrollar todas las fases de un proyecto de tal envergadura y poder alcanzar su correcta finalización me llena de confianza para emprender nuevos caminos.

Personalmente, estoy muy satisfecho con el resultado que se ha obtenido. Creo que he podido dar respuesta a todos los objetivos que me marqué al inicio, que parece ahora tan lejano. No hace falta mencionar que ésta no es más que la primera versión de un proyecto que podría crecer ampliamente, y que, de hecho, me gustaría continuar una vez terminado el grado. Pese a nacer de una situación muy concreta, considero que puede dar respuestas a necesidades de otros usuarios, con lo que resulta viable buscarle una salida comercial en un futuro.

Sinceramente, me llena de orgullo pensar que éste es el proyecto con el que tengo la suerte de cerrar esta etapa, sabiendo que con esfuerzo y dedicación somos capaces de hacer todo lo que nos proponamos.

# Anexo 1: Entregables del proyecto

## Documentación

La carpeta **doc/** contiene los siguientes ficheros:

- **PAC\_FINAL\_mem\_LeonEsquivel\_Sergio.pdf**: Documento con la memoria del proyecto.
- **Anexo1\_DiagramaGantt.pdf**: Diagrama de Gantt que muestra la planificación temporal del proyecto.
- **Anexo1\_BDDModeloER.pdf**: Modelo entidad-relación de la base de datos.
- **Anexo1\_ArduinoUML.pdf**: Diagrama UML con la definición y relación de clases del dispositivo Arduino.
- **Anexo1\_ServidorUML.pdf**: Diagrama UML con la definición y relación de clases del servidor.
- **Anexo1\_WireframesBajaFidelidad.pdf**: Sucesión de pantallas con el diseño compositivo básico de la aplicación.
- **Anexo1\_WireframesAltaFidelidad.pdf**: Sucesión de pantallas con el diseño de interfaz definitivo de la aplicación.
- **Anexo5\_GuiaUsuario.pdf**: Guía de usuario con instrucciones de uso de la plataforma.
- **Anexo6\_LibroEstilo.pdf**: Libro de estilo con la definición del apartado gráfico de la plataforma.

## Proyecto

La carpeta **prj/** contiene los siguientes ficheros:

- **PAC2\_prj\_LeonEsquivel\_Sergio.zip**: Código de la plataforma. Contiene las siguientes carpetas:
  - a. **arduino**: Código del sketch y las clases desarrolladas para el dispositivo.
  - b. **prototipo**:
    - i. **db**: Primera versión de la base de datos en formato SQL.

ii. **server:** Código del servidor. Contiene el prototipo de conexión y las funciones de *back-end* necesarias para realizar el *login* y las solicitudes a Arduino.

c. **app:**

iii. **db:** Base de datos de la aplicación en formato SQL.

iv. **server:** Código del servidor. Contiene todo el *front-end* de la aplicación y la página de descarga así como la lógica de la REST API de servidor y todas las funciones de *back-end*.

v. **apk:** Paquete instalable para dispositivos Android.

## Anexo 2: Código fuente

Algunas de las partes más relevantes del código de la plataforma son:

### Arduino

- **Inicialización y configuración del módulo Ethernet:** Las siguientes líneas de código configuran la IP del dispositivo. La dirección MAC viene definida en la *shield*, y la IP es la dirección que se quiere otorgar al dispositivo dentro de la red local.

```
26 | // MAC/IP configuration for ethernet shield (mac address obtained from  
    | shield's label)  
27 | byte mac[] = { 0x90, 0xA2, 0xDA, 0x00, 0x4D, 0xC1 };  
28 | byte ip[] = { 192, 168, 1, 199 };
```

**Código 2.** Configuración de la dirección IP y MAC del dispositivo – pool.ino

El método *begin* de la clase *Ethernet* recibe como parámetros las dos variables anteriores y configura la conexión de red local.

```
49 | Ethernet.begin(mac, ip); // configures the ethernet shield
```

**Código 3.** Configuración de la *Ethernet shield* – pool.ino

- **Inicialización de la API y la REST API:** Habiendo previamente incluido los ficheros *API.h* y *RESTAPI.h*, se crean dos instancias de dicha clase y se inicializan con el método *begin*. Como se puede comprobar en el siguiente código, la clase *RESTAPI* requiere un apuntador al objeto de la clase *API* para poder interactuar con ella.

```
50 | api.begin(); // starts the API  
51 | restapi.begin(&api); // starts the RESTAPI (needs a pointer to the api to  
    | handle the data)
```

**Código 4.** Inicialización de las clases *API* y *RESTAPI* – pool.ino

- **Creación del servidor web:** El constructor de la clase *WebServer* espera dos parámetros. El primero indica la ruta en la que se ubicarán todas las páginas del servidor, en este caso en la raíz, y el segundo define el puerto en el que creará el servidor.

```
30 | // Webservice initialization
31 | WebServer webserver("", 80);
```

**Código 5.** Creación del servidor web – pool.ino

Las siguientes líneas configuran el servidor web definiendo los puntos de acceso al servidor. En este caso sólo se tratarán las solicitudes que apunten a `http://rutaservidor/api/endpoint`. Cualquier otra ruta hará que la clase *WebServer* devuelva un error *EPIC FAIL*. Como se puede ver, se pasa un apuntador a la función *api\_route* que tratará la solicitud y que necesita estar definido en el ámbito global del proyecto, ya que de lo contrario la librería no podría acceder a dicho método.

```
55 | // Webservice configuration
56 | webserver.setDefaultCommand(&api_route);
57 | webserver.addCommand("api", &api_route);
58 | webserver.begin();
```

**Código 6.** Configuración del servidor web – pool.ino

- **Tratamiento de la solicitud del usuario:** La función *api\_route* recibe toda la información de la solicitud y la rederiva al método *route* de la clase *RESTAPI*, que es la que interactúa internamente con la clase *API* para acceder o modificar los valores necesarios.

```
38 | //Function needed to handle webservice requests. It must be global so the
    | webservice library can reach it
39 | void api_route(WebServer &server, WebServer::ConnectionType type, char
    | *url_tail, bool tail_complete){
40 |     String result = restapi.route(server, type, url_tail);
41 |     server << result;
42 | }
```

**Código 7.** Función de tratamiento de solicitudes REST – pool.ino



- **Envío de datos al servidor:** La función `sendData()` de la clase API es la encargada de generar el código JSON con todas las mediciones que se desean incorporar en la base de datos y tramitar su correspondiente envío al servidor. Esta función se ejecuta en cada paso del bucle, e internamente comprueba si debe establecer la comunicación con el servidor o no.

En primer lugar, merece la pena destacar que se realiza un envío con toda la información del dispositivo cada vez que éste se reinicia manualmente o ante alguna caída de electricidad. Para asegurar que todos los canales tienen la lectura disponible, ésta se lleva a cabo 30 segundos después de su arranque.

```
165 | if (_isFirstConnection){
166 |     if (millis() > 30000){ //First connection available after 30 seconds
    |     from boot so every measure is available
192 |     }
193 | }
```

**Código 8.** Envío de datos: Primera conexión – API.cpp

Cuando ya no se trata de la primera conexión, se comprueba si ha pasado el tiempo del intervalo `webpost` especificado por el usuario. Para ello se valida que haya transcurrido dicho periodo de tiempo desde la hora de la última conexión.

```
194 | // Send data according to webpostInterval
195 | unsigned long webpostInterval = _webpostInterval * 60L * 1000L;
196 |
197 | if(millis() - _lastConnectionTime > webpostInterval) {
238 | }
```

**Código 9.** Envío de datos: Envío periódico – API.cpp

En caso de que cualquiera de las dos condiciones sea válida (primera conexión al servidor o conexión periódica en base a `webpost interval`) se incluye el código que permite generar la estructura JSON a enviar al servidor. A continuación se muestra una porción del código a modo de ejemplo:

```
163 | channels = "";
181 | channels = channels +   "{\"code\": \"Modo\", \"datapoints\": [{\"date\": \" +
    | timestamp + \", \"value\": \" + currentData.mode + \"}]}}\";
```

```
182  
183 postData = postData + "{";  
184 postData = postData +   "\"channels\":[ ";  
185 postData = postData +     channels;  
186 postData = postData +   " ]";  
187 postData = postData + "}";
```

**Código 10.** Envío de datos: Generación de estructura JSON – API.cpp

Este código genera una cadena de texto *channels* con la estructura JSON de todos los canales. Merece la pena mencionar que los de tipo binario, es decir, aquellos que contienen valores de encendido-apagado, se envían siempre que su estado ha cambiado para mostrar el momento preciso de la modificación. Para ello, la creación de su JSON se coloca fuera de la validación del intervalo y se incluye una estructura de datos adicional *\_lastDataSent* que almacena el último valor enviado de cada canal, de forma que es posible saber si el estado ha cambiado respecto al último envío.

El último paso consiste en establecer la comunicación con el servidor para enviar todos los datos recopilados:

```
241 if (client.connect("poolduino.sergileon.com", 80)) {  
242     client.println("POST /api/devices/" + String(DEVICE_ID) + "  
HTTP/1.1");  
243     client.println("Host: poolduino.sergileon.com");  
244     client.println("X-ApiKey: " + String(API_KEY));  
245     client.println("User-Agent: Arduino/1.0");  
246     client.println("Content-Type:application/x-www-form-urlencoded;  
charset=UTF-8");  
247     client.println("Connection: close");  
248     client.print("Content-Length: ");  
249     client.println(PostData.length());  
250     client.println();  
251     client.println(PostData);  
252     client.stop();  
253  
254     _lastDataSent = currentData;  
255 }else{  
256     client.stop();  
257 }
```

**Código 11.** Envío de datos: Comunicación con el servidor – API.cpp

## Servidor

- **Comunicación con Arduino desde el servidor:** La función *Method::run()* recibe la información de la solicitud a realizar. Para contactar con Arduino hace uso de la utilidad *cURL*, tal y como muestra el siguiente código.

```
144 | $ch = curl_init();
145 |
146 | curl_setopt($ch, CURLOPT_URL, $url);
147 | curl_setopt($ch, CURLOPT_CUSTOMREQUEST, $method);
148 |
149 | if($method == "POST" || $method == "PUT"){
150 |     curl_setopt($ch, CURLOPT_POST, true);
151 |     curl_setopt($ch, CURLOPT_POSTFIELDS, $params);
152 | }
153 |
154 | curl_setopt($ch, CURLOPT_TIMEOUT, 10);
155 | curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);
156 |
157 | $output = curl_exec($ch);
158 |
159 | curl_close($ch);
```

**Código 12.** Código de comunicación con Arduino – PoolDuinoAPI.Methods.php

- **Control de solicitudes REST API:** Las siguientes líneas de código permiten tratar la solicitud enviada a la REST API. El método *\$app->get* permite escuchar sólo las peticiones GET dirigidas hacia el *endpoint* indicado en el primer parámetro. La función *Headers::userCheck(\$g\_headers)* comprueba que la solicitud reciba un *token* válido. La sentencia *\$app->request* permite obtener información de la solicitud para tratar los parámetros enviados por *QueryString* (si es una petición GET) o por el cuerpo o *body* (si es una petición POST). Con la sentencia *echo json\_encode(\$result)* se devuelve la información del servidor en formato JSON. Todo este código se sitúa dentro de un bloque *try-catch* de forma que se pueda gestionar cualquier posible error ocasionado.

```
110 | $app->get('/devices/:device_id/channels/:channel_id/', function
    | ($device_id, $channel_id) use ($app) {
111 |     global $g_headers;
112 |     try{
```

```
113     Headers::userCheck($g_headers);
114
115     $req = $app->request();
116     $from = $req->get('from');
117     $to = $req->get('to');
118
119     $channel = Channels::get($device_id, $channel_id);
120     $channel['datapoints'] = Datapoints::listAll($device_id,
121     $channel_id, $from,
122
123     $result = array("result" => $channel);
124     echo json_encode($result);
125 }catch(PoolDuinoAPIException $exception){
126     $error = array("error" => array("message" => $exception-
127     >getMessage(), "code" => $exception->getCode()));
128     echo(json_encode($error));
129 }
130 });
```

**Código 13.** Código de control de solicitud REST API – route.php

- **Uso de excepciones PoolDuinoAPI:** La clase *PoolDuinoAPIException* permite definir una excepción personalizada para la API creada. Los códigos de error se definen como constantes dentro de la clase para no tener que escribir el mensaje y el código a mano:

```
3 class PoolDuinoAPIException extends Exception{
4
5     const DEVICE_DOES_NOT_EXIST = "Device does not exist.";
6     const DEVICE_DOES_NOT_EXIST_CODE = 1001;
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44 }
```

**Código 14.** Fragmento de PoolDuinoAPIException – PoolDuinoAPIException.php

Para lanzar una excepción basta con ejecutar la siguiente línea:

```
54 throw new
    PoolDuinoAPIException(PoolDuinoAPIException::DEVICE_DOES_NOT_EXIST,
    PoolDuinoAPIException::DEVICE_DOES_NOT_EXIST_CODE);
```

**Código 15.** Uso de PoolDuinoAPIException – PoolDuinoAPI.Devices.php

- **Envío de notificaciones:** La clase *Notifications* de *PoolDuinoAPI* contiene el método *send()*, que permite enviar una notificación a un usuario en concreto. Este método de interacción con el cliente puede ser mediante una notificación *push* y/o mediante un mensaje de correo electrónico. En el caso de la notificación *push*, primero consulta los *tokens* de los terminales y luego conecta con el servicio GCM:

```
201 | $registrationIds = array();
202 | while ($row = $g_db->fetchArray($rows)) {
203 |     array_push($registrationIds, $row['token']);
204 | }
205 |
206 | $data = array(
207 |     'title' => $title,
208 |     'message' => $message,
209 |     'notificationId'
210 |     'vibrate' => 1,
211 |     'sound' => 1
212 | );
213 |
214 | $fields = array(
215 |     'registration_ids' => $registrationIds,
216 |     'data' => $data
217 | );
218 |
219 | $headers = array(
220 |     'Authorization: key=' . $cfg_gcm_api_key,
221 |     'Content-Type: application/json'
222 | );
223 |
224 | $ch = curl_init();
225 | curl_setopt( $ch,CURLOPT_URL, 'https://android.googleapis.com/gcm/send' );
226 | curl_setopt( $ch,CURLOPT_POST, true );
227 | curl_setopt( $ch,CURLOPT_HTTPHEADER, $headers );
228 | curl_setopt( $ch,CURLOPT_RETURNTRANSFER, true );
229 | curl_setopt( $ch,CURLOPT_SSL_VERIFYPEER, false );
230 | curl_setopt( $ch,CURLOPT_POSTFIELDS, json_encode($fields) );
231 | $result = curl_exec($ch);
232 | curl_close( $ch );
```

**Código 16.** Envío de notificaciones *push* – PoolDuinoAPI.Notifications.php

En el caso de la notificación por correo electrónico, recupera el *email* del usuario y luego envía el mensaje utilizando el SMTP del servidor:

```
165 | $to = $row['email'];
166 |
167 | $mail = new PHPMailer();
168 |
169 | $mail->IsSMTP();
170 | $mail->CharSet = 'UTF-8';
171 | $mail->SMTPAuth = true;
172 | $mail->Host = $cfg_smtp_host;
173 | $mail->Port = $cfg_smtp_port;
174 | $mail->Username = $cfg_smtp_user;
175 | $mail->Password = $cfg_smtp_pwd;
176 |
177 | $mail->SetFrom($cfg_notif_from, $cfg_notif_from_name);
178 | $mail->AddAddress($to);
179 | $mail->Subject = $title;
180 | $mail->Body = $message;
181 |
182 | $mail->Send();
```

**Código 17.** Envío de notificaciones por *email* – PoolDuinoAPI.Notifications.php

## Cliente

- **Acceso a información del dispositivo:** La librería Apache Cordova permite acceder a información y funcionalidades del dispositivo desde JavaScript. En este caso, nos permite conocer datos del dispositivo. Por otro lado, el *plugin* oficial de PhoneGap GCMPlugin nos permite registrar el terminal para su uso con GCM:

```
1480 | function initDevice(){
1481 |     if(g_bFromPhoneGap){
1482 |         document.addEventListener('deviceready', function() {
1483 |             window.plugins.GCM.register("269844236197", "GCM_Event",
GCM_Success, GCM_Fail);
1484 |             g_deviceInfo.model = device.model;
1485 |             g_deviceInfo.platform = device.platform;
1486 |             g_deviceInfo.uuid = device.uuid;
1487 |             g_deviceInfo.version = device.version;
1488 |         }, false);
1489 |     }
1490 | }
```

**Código 18.** Acceso a información del dispositivo y registro para GCM – app.js

## Anexo 3: Librerías/código externo utilizado

Las librerías de terceros que se han utilizado en este proyecto son:

### Arduino

- **Ethernet y EthernetUDP:** La primera habilita la conexión a Internet para la placa Arduino y la segunda habilita el envío y recepción de mensajes por UDP. Ambas son librerías que se suministran junto al IDE. La página oficial de la librería es <http://arduino.cc/en/reference/ethernet>. El uso de la librería Ethernet es indispensable para poder acceder al sistema remotamente, y el de EthernetUDP es necesario para poder realizar la llamada de sincronización de la hora por NTP.
- **Webduino:** Librería que permite crear un servidor web en Arduino. La página oficial del proyecto es <https://code.google.com/p/webduino/>. Para su uso en el proyecto ha sido necesaria modificar la función *WebServer::dispatchCommand* para que pueda tratar direcciones especificadas en formato REST, ya que de lo contrario esperaba recibir la ruta especificada íntegramente con *QueryString*. El código de la función queda como se muestra a continuación:

```
111 | bool WebServer::dispatchCommand(ConnectionType requestType, char *verb,
112 |     bool tail_complete)
113 | {
114 |     // if there is no URL, i.e. we have a prefix and it's requested
115 |     // without a
116 |     // trailing slash or if the URL is just the slash
117 |     if ((verb[0] == 0) || ((verb[0] == '/') && (verb[1] == 0)))
118 |     {
119 |         m_defaultCmd(*this, requestType, "", tail_complete);
120 |         return true;
121 |     }
122 |     // if the URL is just a slash followed by a question mark
123 |     // we're looking at the default command with GET parameters passed
124 |     if ((verb[0] == '/') && (verb[1] == '?'))
125 |     {
126 |         verb+=2; // skip over the "/"? part of the url
127 |         m_defaultCmd(*this, requestType, verb, tail_complete);
128 |     }
129 | }
```

```
127     return true;
128 }
129 // We now know that the URL contains at least one character. And,
130 // if the first character is a slash, there's more after it.
131 if (verb[0] == '/')
132 {
133     char i;
134     char *qm_loc;
135     int verb_len;
136     int qm_offset;
137     // Skip over the leading "/", because it makes the code more
138     // efficient and easier to understand.
139     verb++;
140
141     // Looks for a command which matches the verb
142     bool bFound = false;
143     int n = 0;
144     int n_command = -1;
145
146     for (i = 0; i < m_cmdCount; ++i){
147         n=0;
148         if (strncmp(m_commands[i].verb, verb, strlen(m_commands[i].verb))
149             == 0){
150             if(strlen(m_commands[i].verb) > n){
151                 n_command = i;
152                 n = strlen(m_commands[i].verb);
153             }
154         }
155
156         if(n_command != -1){
157             qm_offset = 0;
158
159             m_commands[n_command].cmd(*this, requestType,
160                 verb + n + qm_offset,
161                 tail_complete);
162             return true;
163         }else{
164             return false;
165         }
166     }
167     return false;
168 }
```

Código 19. Método WebServer::dispatchCommand – WebServer.cpp



Por otra parte, la librería se proporciona en un archivo con extensión **.h** que incluye tanto la definición de las cabeceras como el código de los métodos. Se aprovecha para separar el código incluyéndolo en un archivo con extensión **.cpp**, de acuerdo con las buenas prácticas de desarrollo de librerías.

- **EEPROMex**: Permite escribir valores complejos en la EEPROM del Arduino. La página oficial del proyecto es <http://thijs.elenbaas.net/2012/07/extended-EEPROM-library-for-arduino/>. Esta librería es necesaria para guardar los valores de los parámetros del sistema de forma que, ante un corte de alimentación inesperado, arranque con la última configuración establecida.

## Front-end

- **Apache Cordova**: Librería que permite acceder a las funcionalidades del dispositivo móvil desde JavaScript para crear aplicaciones nativas de forma sencilla. La página oficial del proyecto es <http://cordova.apache.org/>.
- **Bootstrap**: Librería que permite simplificar el diseño de la estructura de la página de forma responsiva para que se adapte perfectamente a las dimensiones de la pantalla. La página oficial del proyecto es <http://getbootstrap.com/>.
- **Bootstrap DatePicker**: Librería que permite insertar un *widget* de selección de fecha para un campo HTML, mejorando la usabilidad de la aplicación. La página oficial del proyecto es <http://www.eyecon.ro/bootstrap-datepicker/>.
- **Bootstrap Timepicker**: Librería que permite insertar un *widget* de selección de hora para un campo HTML, mejorando la usabilidad de la aplicación. La página oficial del proyecto es <http://jdewit.github.io/bootstrap-timepicker/>.
- **FontAwesome**: Librería que permite insertar fuentes iconográficas en la páginas con sencillos estilos CSS. Al ser vectoriales pueden escalarse a cualquier dimensión sin perder definición, muy útil en entornos donde la resolución de pantalla es tan variable. La página oficial del proyecto es <http://fontawesome.io/>.

- **Flot:** Librería para jQuery que permite crear gráficas de forma dinámica. La página oficial del proyecto es <http://www.flotcharts.org/>.
- **jQuery:** Biblioteca para simplificar el desarrollo en cliente con JavaScript. La página oficial del proyecto es <http://jquery.com/>. Esta librería ofrece la posibilidad de realizar llamadas AJAX de forma ágil y sencilla, permitiendo actualizar el contenido sin tener que refrescar la página y mejorando, por lo tanto, la usabilidad.
- **jQuery UI:** Librería para jQuery que proporciona un conjunto de elementos de interfaz, efectos, *widjets* y temas predefinidos. La página oficial del proyecto es <http://jqueryui.com/>.
- **TouchSwipe:** *Plugin* para jQuery que permite tratar interacciones táctiles complejas como deslizamientos y entrada múltiple. La página oficial del proyecto es <http://labs.rampinteractive.co.uk/touchSwipe/demos/>.

## *Back-end*

- **Slim:** *Framework* para PHP que permite crear una API en formato REST de forma fácil e intuitiva. La página oficial del proyecto es <http://www.slimframework.com/>.
- **PHPMailer:** Clase que habilita el envío de correos electrónicos a través de un servidor SMTP. La página oficial del proyecto es <http://phpmailer.worxware.com/>.

## Anexo 4: Capturas de pantalla

A continuación se muestran algunas capturas de pantalla del proceso de desarrollo.

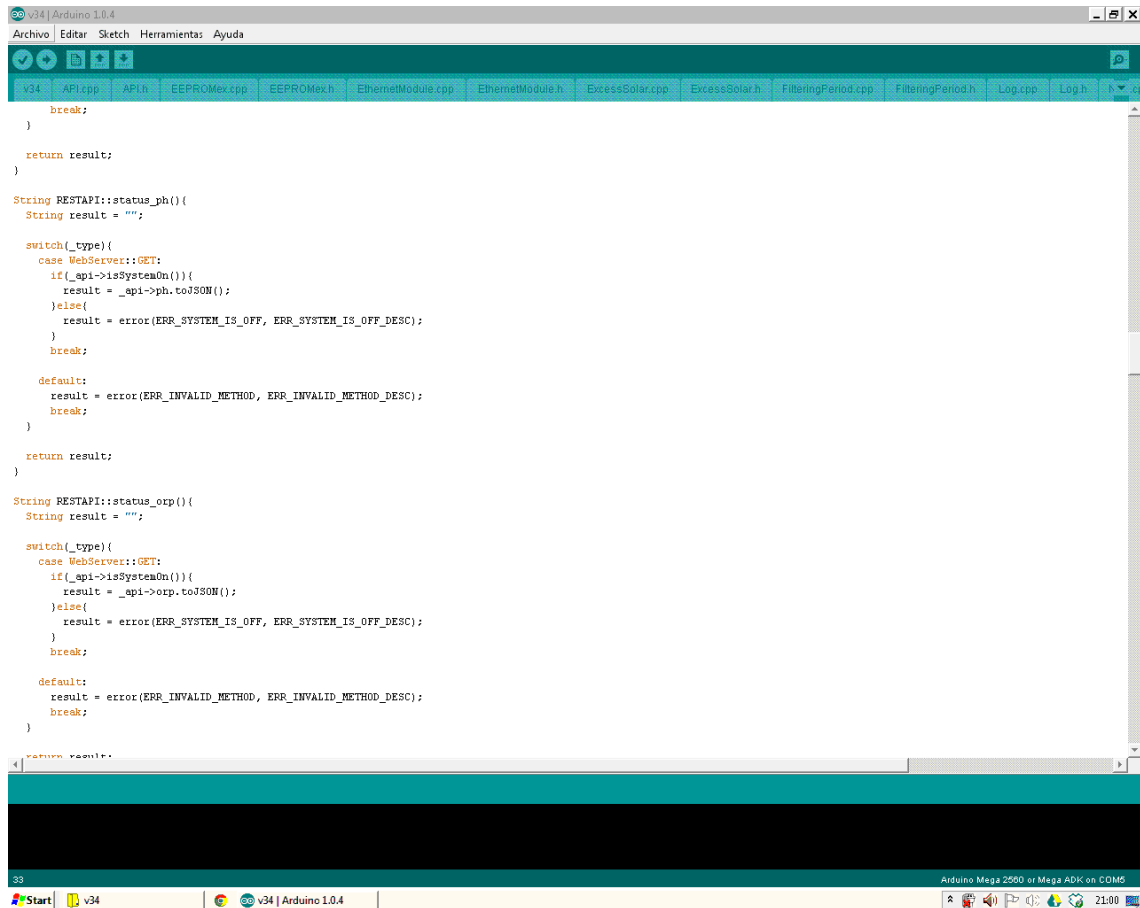


Figura 54. Arduino IDE accediendo vía escritorio remoto

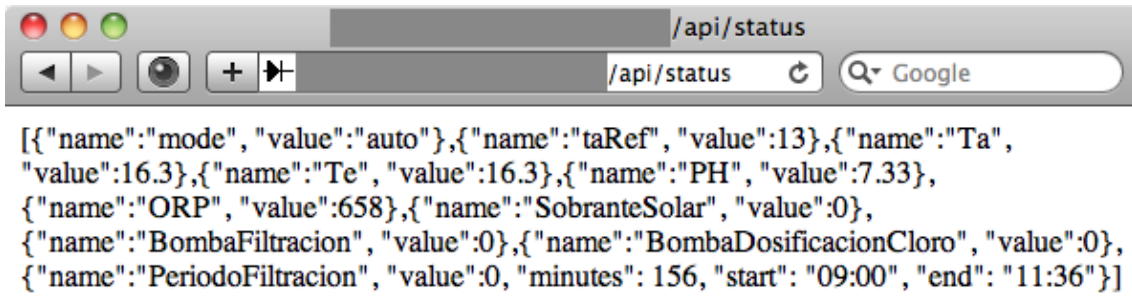


Figura 55. Acceso a Arduino mediante HTTP y REST



Figura 56. Acceso a Arduino mediante el prototipo creado en la fase 1

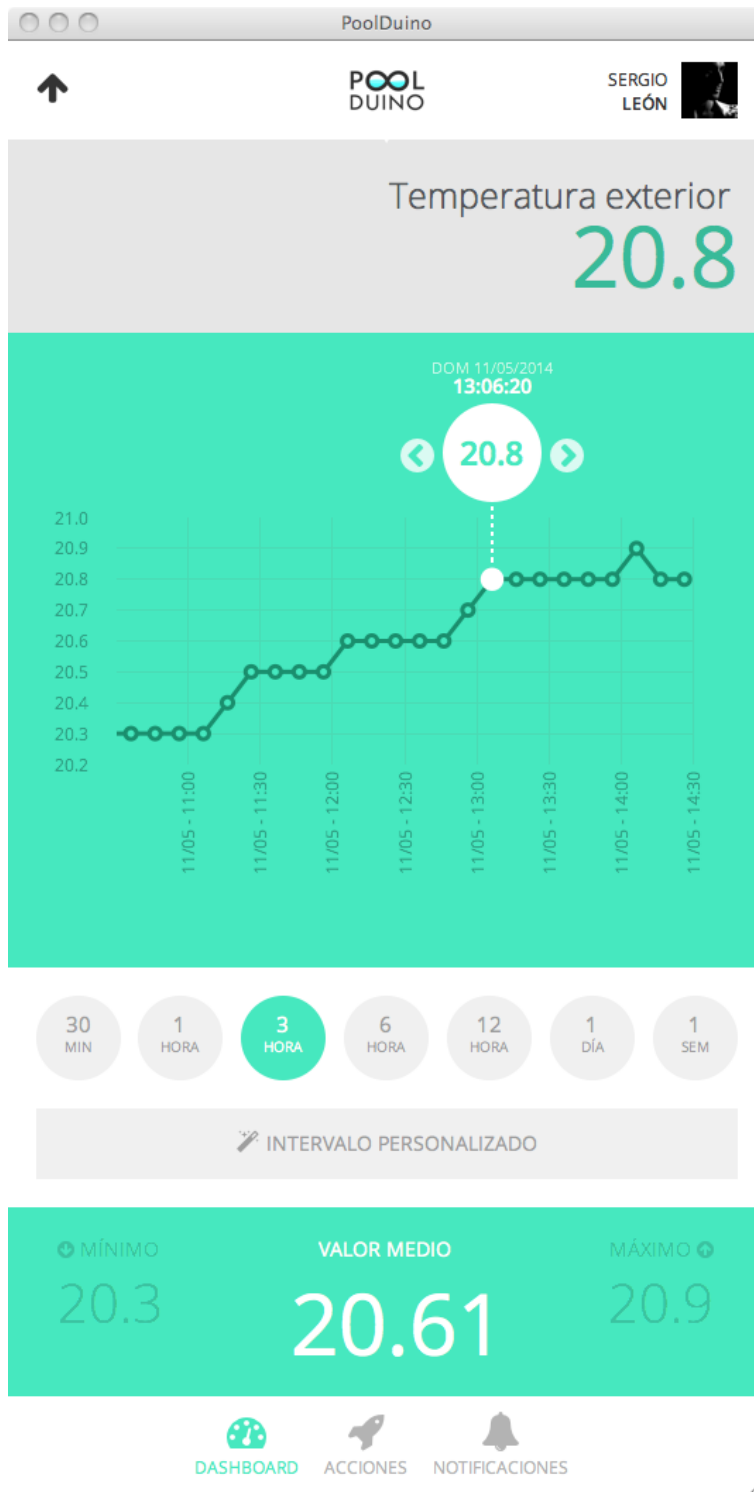


Figura 57. Información dinámica del canal 'Temperatura exterior'

## Anexo 5: Guía de usuario

El fichero Anexo5\_GuiaUsuario.pdf contiene una versión de mejor legibilidad de la guía de usuario para el proyecto desarrollado.



Figura 58. Guía de usuario: *Login* y Estructura de la aplicación

### 3. Usuario

Al pulsar sobre el icono del usuario en la cabecera se despliegan las opciones referentes a su cuenta.



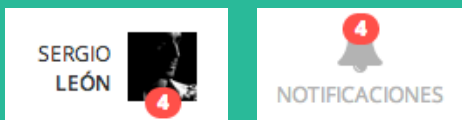
A continuación explicamos en qué consiste cada opción:

**Ver notificaciones** 🔔 conduce al usuario al apartado de notificaciones.

**Bloquear sesión** 🔒 concluye la sesión del usuario conduciéndolo a la pantalla de *login*, pero en este caso sólo deberá introducir la contraseña para autenticarse. Además, en caso de acceder desde un dispositivo móvil, éste seguirá recibiendo notificaciones *push*.

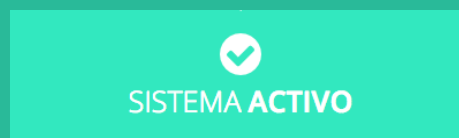
**Cerrar sesión** ➔ concluye la sesión del usuario y lo conduce a la pantalla de *login*. A diferencia de la opción anterior, la sesión se destruye completamente y tendrá que volver a insertar la combinación de usuario/contraseña para volverse a autenticar. Además, el dispositivo dejará de recibir notificaciones *push*.

Por otra parte, el usuario puede comprobar rápidamente el número de notificaciones por tratar gracias a los indicadores visuales habilitados para ello, que se encuentran tanto en la cabecera como en el pie de página, junto a la opción *Notificaciones*.

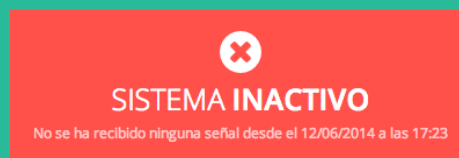


### 4. Dashboard

Esta sección muestra el resumen del estado actual de todos los canales del dispositivo. El primer bloque muestra el estado global del sistema para indicar si está funcionando correctamente:



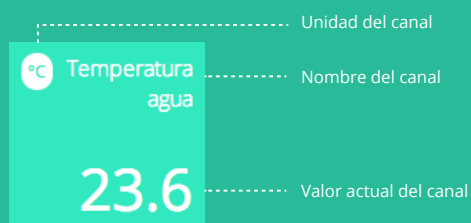
Si el dispositivo no envía datos durante un periodo de tiempo, el bloque muestra un mensaje del siguiente estilo:



El botón **Actualizar** permite refrescar la página mostrando cualquier nuevo valor que se haya recibido en el sistema con fecha posterior a la última actualización.



La lista de canales está representada por bloques con la siguiente estructura:

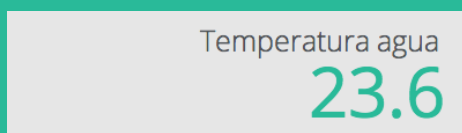


Cada bloque, como muestra la figura anterior, presenta el nombre del canal, su valor actual y su unidad, si dispone de ella. Al presionar sobre la caja se accede a la pantalla de detalle del canal con información específica sobre el mismo.

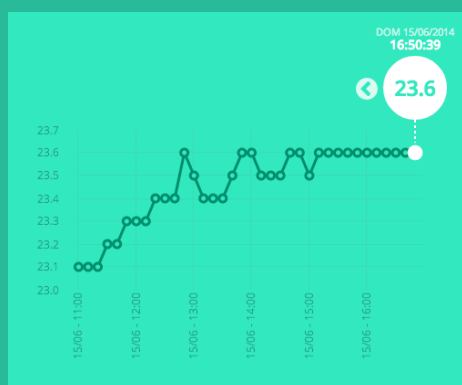
Figura 59. Guía de usuario: Usuario y Dashboard

## 5. Detalle del canal

Esta sección muestra información detallada sobre el canal seleccionado. La cabecera muestra el nombre del canal y el valor actual.



A continuación se incluye una gráfica con la evolución de las mediciones tomadas a lo largo del tiempo:



El eje vertical muestra el rango de valores que ha tomado la medición para el intervalo seleccionado, y el eje horizontal indica el tiempo en el que se han tomado las mediciones.

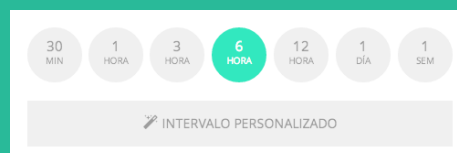
Cada punto de la gráfica muestra información del valor y el momento en el que se tomó la medición:



Se puede navegar por los puntos de la gráfica utilizando los cursores que se muestran a ambos lados del valor seleccionado. El primer y último punto no permiten navegar hacia la izquierda y derecha respectivamente.

También es posible navegar por la gráfica pasando el cursor por encima de los diferentes puntos o moviéndonos con las teclas de dirección ◀ y ▶ del teclado. Otra posible forma de interactuar con la gráfica es pulsando directamente sobre los puntos, de forma que es accesible cuando se consulta a través de dispositivos móviles.

La selección del intervalo se lleva a cabo en el siguiente bloque:



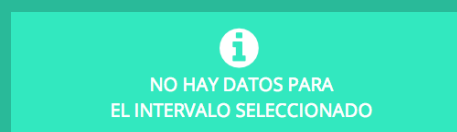
30 MIN 1 HORA 3 HORA 6 HORA 12 HORA 1 DÍA 1 SEM  
INTERVALO PERSONALIZADO

Como podemos ver, existen intervalos predefinidos y la opción de definir manualmente el periodo de tiempo que nos interesa. La opción seleccionada se resalta en verde.



DEFINIR INTERVALO  
INICIO  
15/06/2014  
17:00:15  
FIN  
15/06/2014  
17:00:15  
CANCELAR ACEPTAR

La opción **Intervalo personalizado** permite seleccionar el periodo manualmente mediante el formulario anterior. En caso de seleccionar un intervalo para el que no se disponen datos, se muestra el siguiente mensaje:



Por último, se incluye un resumen del canal con el valor mínimo, el máximo y la media para el intervalo escogido.



Figura 60. Guía de usuario: Detalle del canal

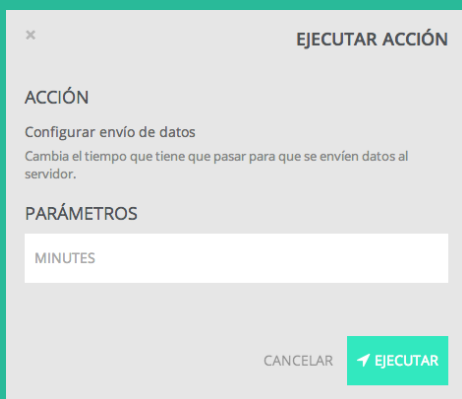


## 6. Acciones

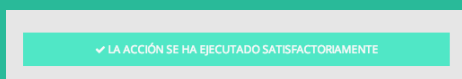
Esta sección muestra las diferentes acciones que se pueden ejecutar sobre el dispositivo.



Al presionar sobre el botón se abre una ventana que ofrece información adicional de la acción y permite la entrada de los parámetros necesarios para su correcto funcionamiento.



Si presionamos **Ejecutar** se valida el parámetro introducido, si existe alguno, y se lleva a cabo la acción especificada en el dispositivo. El resultado se muestra por pantalla:



La ejecución, sin embargo, está restringida sólo a aquellos usuarios que disponen de permisos suficientes.



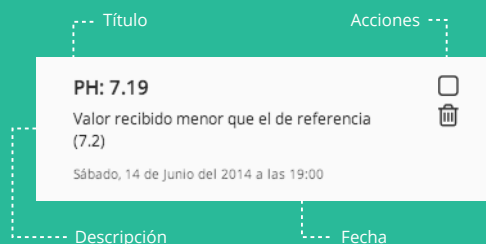
## 7. Notificaciones

Esta sección muestra el listado de notificaciones que el usuario ha recibido en base a las reglas que ha definido previamente en la base de datos. La definición de reglas todavía no es accesible a través de la aplicación.



En el ejemplo anterior se muestran dos notificaciones enviadas automáticamente en caso que se reciban mediciones del canal PH por debajo de 7.2.

Cada notificación presenta la siguiente estructura:



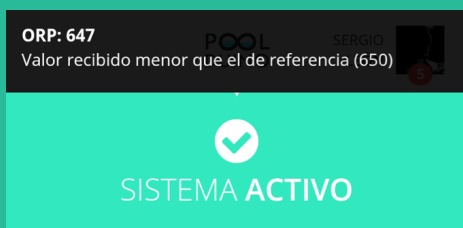
Por defecto, las notificaciones aparecen en estado de no tratadas (*blanco*). Al presionar sobre el bloque, la notificación alterna entre tratada (*gris*) y no tratada, y se actualiza automáticamente el contador asociado.

En la versión de escritorio aparece el icono que permite eliminar la notificación seleccionada. Accediendo a través del dispositivo se oculta dicho botón con el objetivo de simplificar la interfaz. En este caso, el procedimiento para borrar una notificación se lleva a cabo deslizando el dedo hacia la derecha (*swipe right*).

Figura 61. Guía de usuario: Acciones y Notificaciones

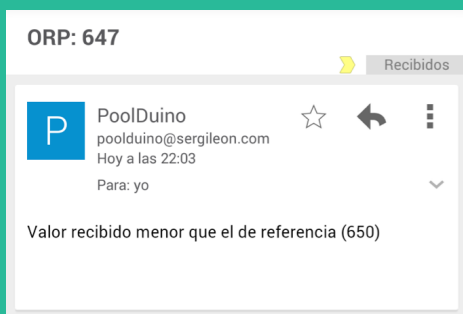
## 7. Notificaciones

Cuando accedemos a **PoolDuino** a través de la aplicación para dispositivos móviles, al recibir una notificación *push* mientras tenemos la aplicación abierta se mostrará el mensaje por pantalla, tal y como muestra la siguiente imagen:



Al presionar sobre este bloque emergente, que desaparece en dos segundos, se accede directamente a la lista de notificaciones.

Para todos aquellos usuarios que no disponen de un dispositivo móvil compatible con la aplicación se ha desarrollado un sistema de notificaciones alternativo basado en el envío de correos electrónicos, de forma que aún puedan recibir las alertas que les interesen. Al definir una regla se puede indicar qué método o combinación de métodos se usarán para enviar el aviso al usuario.



En la imagen vemos un correo electrónico con la misma información que proporciona una notificación *push*.

## 8. Definición de reglas

Pese a que este apartado no es accesible a través de la aplicación, tiene una gran relevancia en el proyecto. La definición de reglas permite automatizar el envío de notificaciones cuando se reciben valores del dispositivo que cumplen las condiciones definidas por el usuario.

Las reglas son únicas para cada usuario, y se definen en la tabla *tb\_channel\_triggers* con la siguiente estructura:

user_id	channel_id	operator	value	notif_mode
1	4	<	650	N/M
1	3	<	7.2	N/M
1	3	>	7.6	N/M
1	4	>	750	N/M

A continuación explicamos el contenido de cada campo:

**user\_id:** Identificador del usuario en el sistema

**channel\_id:** Identificador del canal

**operator:** Operador que se desea incluir en la condición. Los valores posibles son =, < y >.

**value:** Valor de referencia que se comparará con el valor que se pretende añadir en el sistema.

**notif\_mode:** Método de notificación. Los valores posibles son N (notificación *push*) y M (*mail*). Se puede definir uno solo o incluir varios separados por el carácter |.

Así pues, en la imagen podemos ver cómo el usuario 1 ha definido 4 reglas, dos de ellas para el canal 3 (pH) y otras dos para el canal 4 (ORP). Ambas validan los umbrales inferior y superior que se estiman necesarios (en este caso, 650 y 750 para el caso del canal ORP y 7.2 y 7.6 para el pH).

Una vez definido este registro, cada vez que se reciba un nuevo valor se comprobará si éste cumple la condición establecida y creará la notificación asociada en caso afirmativo.

Teniendo en cuenta que un usuario puede haber iniciado sesión en múltiples dispositivos al mismo tiempo, las notificaciones *push* se envían a todos los terminales que se hayan registrado en la plataforma para el usuario que ha definido la regla.

Figura 63. Guía de usuario: Notificaciones y Definición de reglas

## Anexo 6: Libro de estilo

El fichero Anexo6\_LibroEstilo.pdf contiene una versión de mejor legibilidad del libro de estilo desarrollado para este proyecto.

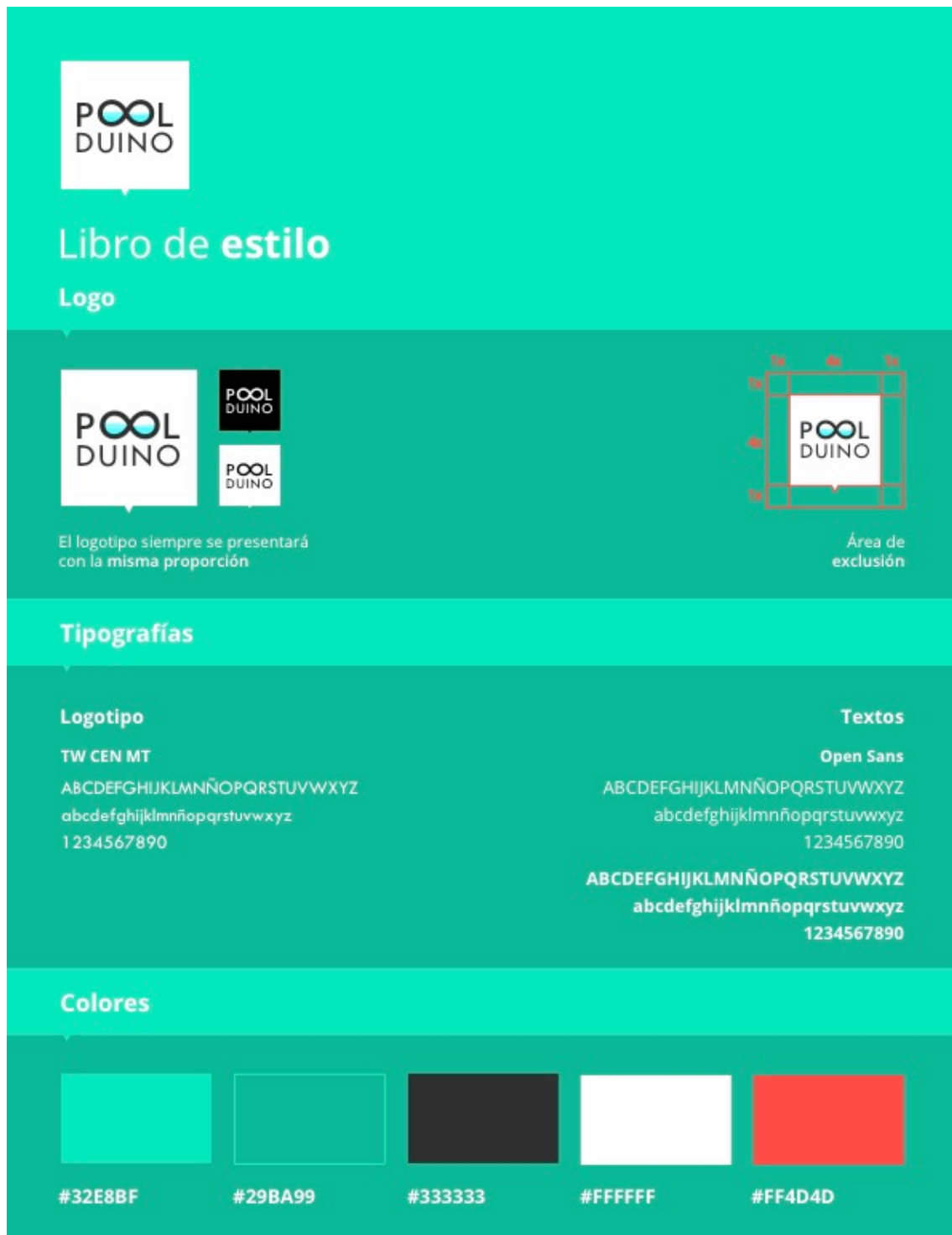


Figura 62. Libro de estilo

## Anexo 7: Glosario

**AJAX:** Acrónimo de *Asynchronous JavaScript And XML*. Permite crear aplicaciones interactivas posibilitando realizar cambios en las páginas sin tener que recargarlas.

**Apache:** Servidor web HTTP de código abierto.

**API:** Acrónimo de *Application Programming Interface*. Conjunto de funciones y métodos que definen una interfaz de comunicación programática con el *software*.

**APK:** Siglas de *Application PacKage File*. Formato que poseen los paquetes instalables en Android.

**Arduino:** Plataforma de *hardware* de código abierto que permite interactuar con elementos electrónicos.

**Back-end:** Código del servidor que procesa la información solicitada por el usuario.

**Backup:** Copia de seguridad.

**Biblioteca:** Colección de funciones o métodos codificados en un lenguaje de programación concreto. La utilizan otros procedimientos del programa para evitar reescribir código o para abstraerle de su funcionamiento interno.

**Bootstrap:** Colección de herramientas creada por Twitter para crear sitios o aplicaciones web responsivas.

**CSS:** Siglas de *Cascading Style Sheets*. Permite definir el aspecto gráfico que debe tener un documento HTML.

**cURL:** Herramienta para transferir datos usando diferentes protocolos a través de un terminal mediante línea de comandos.

**Dashboard:** Interfaz con la que se puede administrar un sistema y que, normalmente, muestra información resumida para ofrecer un enfoque global.

**DCU:** Siglas de Diseño Centrado en el Usuario. Procedimiento que dirige el diseño de un producto teniendo en cuenta las necesidades concretas de los usuarios que lo harán servir.

**Diagrama de Gantt:** Representación gráfica que permite mostrar de forma intuitiva la planificación temporal de un proyecto.

**Diseño responsivo:** Diseño que se adapta a la resolución de pantalla del dispositivo desde el que se accede a la información, redimensionando o redistribuyendo sus elementos para mostrar una versión optimizada de la página.

**Dropbox:** Servicio de almacenamiento de archivos *cloud* o en la nube.

**End point:** Dirección del recurso que se consulta desde una REST API.

**Ethernet:** Estándar para tecnologías de redes de área local.

**FontAwesome:** Fuente icónica vectorial ideal para su uso en páginas web.

**Framework:** Conjunto de utilidades que permiten facilitar el proceso de desarrollo o implementación de una aplicación.

**Front-end:** Parte del código que corresponde a la interfaz de interacción con el usuario.

**FTP:** Siglas de *File Transfer Protocol*. Protocolo de red que permite la transferencia de archivos entre ordenadores.

**Hit area:** Zona de la página en la que el usuario puede pulsar.

**HTML:** Siglas de *HyperText Markup Language*. Lenguaje de marcado que permite definir la estructura de una página web.

**IDE:** Acrónimo de *Integrated Development Environment*. Entorno de programación que ofrece las herramientas necesarias para desarrollar programas basados en un lenguaje o en varios.

**Internet of things:** También conocido como Internet de las cosas, consiste en el establecimiento de una red de objetos interconectados que ofrecen datos en tiempo real.

**Interfaz:** Medio con el que el usuario puede comunicarse con un ordenador. También se puede entender como el medio con el que un programa interactúa con otro.

**iOS:** Sistema operativo de los dispositivos móviles de Apple.

**IP:** Siglas de *Internet Protocol*. Etiqueta numérica que identifica de forma única a un equipo dentro de una red que utilice este protocolo.

**JavaScript:** Lenguaje de programación interpretado utilizado mayoritariamente en combinación con HTML para generar páginas web interactivas.

**jQuery:** Biblioteca diseñada para simplificar la programación en cliente con JavaScript.

**JSON:** Acrónimo de *JavaScript Object Notation*. Formato de intercambio de datos.

**Librería:** véase *Biblioteca*.

**Modelo ER o Entidad-Relación:** Herramienta que permite representar gráficamente e intuitivamente las entidades y relaciones presentes en un sistema de información.

**MySQL:** Sistema de gestión de bases de datos relacionales.

**Notificación push:** Tipo de notificación en la que es el servidor el que inicia la comunicación con el dispositivo del usuario sin que éste deba tener ninguna aplicación abierta ni estar solicitando datos al servidor de forma constante.

**NTP:** Siglas de *Network Time Protocol*. Protocolo que permite la sincronización del reloj de un sistema informático.

**Open-source:** Programa cuyo código es abierto y por lo tanto accesible para su consulta y modificación.

**ORP:** Siglas de *Oxidation Reduction Potencial*. Permite expresar la calidad y salubridad del agua.

**pH:** Medida que expresa la acidez o basicidad de un sistema.

**Phonegap:** Framework propiedad de Adobe para el desarrollo de aplicaciones móviles utilizando JavaScript, HTML5 y CSS3.

**PHP:** Siglas de *PHP: Hypertext Preprocessor*. Lenguaje de programación de servidor diseñado para el desarrollo de páginas con contenido dinámico.

**Plug-in o plugin:** Aplicación que se conecta con otra para ofrecerle una funcionalidad adicional específica.

**QueryString:** Cadena de consulta.

**REST:** Acrónimo de *REpresentational State Transfer*. Técnica de arquitectura de desarrollo web basada en el estándar HTTP.

**RTC:** Siglas de *Real Time Clock*. Reloj de un ordenador que le permite mantener la fecha y hora.

**Servicio cloud:** Servicio que se ofrece a través de Internet.

**Servidor cloud:** Servidor que puede ajustar sus recursos dinámicamente.

**Shield:** Placa que puede conectarse encima del Arduino y que extiende sus funcionalidades.

**Spinner:** Icono animado que indica al usuario que una petición ha sido enviada y se está procesando.

**SSL:** Siglas de *Secure Sockets Layer*. Protocolo criptográfico que permite establecer una comunicación segura entre dos máquinas de una red.

**Store:** Tienda o repositorio de los dispositivos móviles desde la que se pueden instalar o actualizar aplicaciones. En Apple se trata de la App Store y en Android se conoce como Google Play.

**Target:** Público objetivo al que se dirige un producto o servicio.

**UML:** Siglas de *Unified Modeling Language*. Permite describir gráficamente los métodos o procesos de un sistema y cómo se interrelacionan entre ellos.

**Wireframe:** Es un esquema, más o menos detallado, del diseño o estructura de una página o producto.

## Anexo 8: Bibliografía

En este apartado se incluyen los recursos utilizados durante el transcurso del proyecto:

Adobe PhoneGap. *PhoneGap API Documentation*

[http://docs.phonegap.com/en/edge/guide\\_platforms\\_index.md.html](http://docs.phonegap.com/en/edge/guide_platforms_index.md.html)

Alias.IO. 2013. *How to store passwords safely with PHP and MySQL*

<http://alias.io/2010/01/store-passwords-safely-with-php-and-mysql/>

Arduino. *Reference*

<http://arduino.cc/en/Reference/HomePage>

Bootstrap. *Getting started*

<http://getbootstrap.com/getting-started/>

Bootstrap TimePicker. *TimePicker for Twitter Bootstrap*

<http://jdewit.github.io/bootstrap-timepicker/>

Bootstrap DatePicker. *DatePicker for Bootstrap*

<http://www.eyecon.ro/bootstrap-datepicker/>

Dev Pro. 2013. *5 Best Practices for Better RESTful API Development*

<http://devproconnections.com/web-development/restful-api-development-best-practices>

Evrything. *API Reference*

<http://dev.evrythng.com/documentation/api>

FontAwesome. *Get Started with Font Awesome*

<http://fontawesome.io/get-started/>

ioBridge. *Documentation*

<http://connect.iobridge.com/docs/>

jQuery. *jQuery API Documentation*

<http://api.jquery.com/>

jQuery UI. *jQuery UI API Documentation*

<http://api.jqueryui.com/>

MySQL. *MySQL 5.5 Reference Manual*

<http://dev.mysql.com/doc/refman/5.5/en/index.html>

PHP. *Documentation*

<http://www.php.net/docs.php>

Slim. *Slim Framework Documentation*

<http://docs.slimframework.com/>

ThingSpeak. *Documentation*

<https://thingspeak.com/docs>

Twitter. *REST API v1.1 Resources*

<https://dev.twitter.com/docs/api/1.1>

Vinay Sahni. 2013. *Best Practices for Designing a Pragmatic RESTful API*

<http://www.vinaysahni.com/best-practices-for-a-pragmatic-restful-api>

W3Schools. *CSS3 Introduction*

[http://www.w3schools.com/css/css3\\_intro.asp](http://www.w3schools.com/css/css3_intro.asp)

W3Schools. *HTML5 Introduction*

[http://www.w3schools.com/html/html5\\_intro.asp](http://www.w3schools.com/html/html5_intro.asp)

Xively. *Xively REST API*

<https://xively.com/dev/docs/api/>



## Anexo 9: Vita

Sergio León Esquivel, nacido el 1989 en Sabadell. Desde muy joven se interesa por el desarrollo de *software* y el diseño gráfico. Gran entusiasta del minimalismo y la simplicidad, y amante incondicional del teatro y la danza. Con la entrega de este proyecto finaliza el Grado Multimedia impartido por la Universitat Oberta de Catalunya. Espera con ganas el inicio de nuevos proyectos relacionados con el mundo de los videojuegos, otra de sus grandes pasiones.