ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADO EN INGENIERÍA DEL SOFTWARE

SISTEMA DE MONITORIZACIÓN BASADO EN PLATAFORMAS ARDUINO, ANDROID Y CLOUD

(Monitoring System Based on Arduino, Android and Cloud Platforms)

Realizado por
Rafael Rodríguez Ordóñez

Tutorizado por
Luis Manuel Llopis Torres

Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, DICIEMBRE 2015

Fecha defensa:
El Secretario del Tribunal

Resumen:

El objetivo de este trabajo fin de grado consiste en la integración de las plataformas Arduino, Android y la Cloud. Con el fin de alcanzar dicho objetivo, se estudiarán dichas tecnologías y se implementará un sistema de monitorización que permita integrar estas tres tecnologías.

Este sistema de monitorización consistirá en visualizar en el dispositivo móvil Android la información recogida por dos sensores (temperatura y luminosidad) conectados a la plataforma Arduino a través de una conexión dentro la misma red WiFi. Para llevar a cabo este proceso la plataforma Arduino llevará acoplado un dispositivo WiFi (shield).

En este sistema se desarrollará dos aplicaciones: Android y Web. La aplicación Android almacenará la información obtenida por la monitorización en una base de datos establecida en la plataforma Cloud de Google mediante servicios web (Google Cloud Endpoints). La aplicación Web estará alojada en Google App Engine junto con la base de datos (DataStore). Esta aplicación Web consistirá en el tratamiento de la información del sistema.

Palabras claves: Arduino, Android, Cloud

Abstract:

The objective of this Final Project is the integration of the Arduino, Android and Cloud platforms. In order to achieve this objective, these technologies will be studied and will be implemented a monitoring system that would integrate the three technologies.

This monitoring system will display on the Android device information collected by two sensors (temperature and luminosity) connected to the Arduino platform through a WiFi connection within the same network. To carry out this process the Arduino platform will have a WiFi (shield) device coupled.

In this system two applications will be developed: Android y Web. The Android application stores the information obtained by monitoring a database established on the Google Cloud platform using web services (Google Cloud Endpoints) The Web application will be hosted on Google App Engine with the database (DataStore). This Web application will consist in the information processing of system.

Keywords: Arduino, Android, Cloud

Agradecimientos

Primero agradecer a mis padres por el apoyo incondicional y gran esfuerzo que han hecho por darme la oportunidad de estudiar. También agradecer a todos los componentes de mi familia por apoyarme y creer en mí, en especial, a mi primo Jorge "Señor Soto" por ser mi otro hermano mayor y por los grandes momento de diversión.

Agradecer a mi tutor y director Luis Manuel Llopis por enseñarme en estos años y confiar en mí para realizar este trabajo.

Agradecer a los amigos que he conocido en el plan antiguo, de mi facultad y de otras, por apoyarme en los buenos y malos momentos tanto dentro como fuera del ámbito universitario. Agradecer a los amigos del plan nuevo por volver a creer en el compañerismo.

Agradecer a mis colegas de siempre, por aguantarme estos años y ayudarme a disfrutar de esta etapa de mi vida.

Por último, y el más importante, agradecer a mi hermano Jorge por estar ahí en todo momento, por engañarme a estudiar esta carrera, por ser como eres y sobre todo por tu gran ayuda para llegar a este momento.

Índice

Índice de Figuras

1. Introducción

La evolución de la tecnología en los últimos años ha crecido de manera exponencial desde la aparición de Internet y los primeros móviles Smartphones, pero esta evolución ha sido más rápida gracias a los sistemas electrónicos o microcontroladores.

1.1 Motivación

La idea de este proyecto surgió por varias circunstancias. Una de ellas fue que me forzaría a aprender tecnologías nuevas que fueran útiles para mi futuro profesional, como es la Cloud Computing. Además, profundizar en otras tecnologías ya vistas durante estos años de estudios, como Android, la plataforma Arduino, la ingeniería Web, entre otras.

Otro motivo fue que me permitiría crear un sistema real donde se utilizaría componentes tanto hardware como software, el cual podría abrirme en otras tecnologías como puede ser la domótica.

1.2 Objetivos TFG

El objetivo principal es diseñar un sistema de monitorización con el fin de integrar un módulo hardware Arduino, un dispositivo móvil con sistema operativo Android y un servidor web alojado en la plataforma Cloud de Google. Dicho sistema contará con una aplicación Android y una aplicación Web.

La aplicación Android consistirá en obtener información de dos sensores diferentes, uno de temperatura y otro de luminosidad, los cuales estarán conectados al módulo Arduino. Esta información de los sensores será enviada a la aplicación desde el servidor Arduino a través de una red Wifi para su posterior monitorización o almacenamiento. Para llevar a cabo este proceso se necesitará un dispositivo Wifi integrado al modulo Arduino para establecer una conexión TCP/IP entre el modulo Arduino y la aplicación móvil.

Esta aplicación Android almacenará automáticamente la información en una base de datos alojada en la plataforma Cloud mediante servicios web proporcionados por Google Cloud Endpoints.

La aplicación Web permite gestionar la información almacenada en el servidor web mediante Google App Engine. En la aplicación podrá configurar condiciones para la monitorización.

1.3 Tecnologías Utilizadas

Durante el desarrollo de este trabajo se han utilizado varias plataformas de software y hardware libre. Las principales tecnologías utilizadas son Arduino, Android y Cloud, pero también ha sido necesario utilizar protocolos o servicios para crear una comunicación entres dichas plataformas, como el protocolo TCP/IP o los servicios Web. A continuación, se detalla las características de las plataformas utilizada para realizar este trabajo.

1.3.1 Arduino

Arduino es una plataforma de hardware libre basada en una placa con un microcontrolador y un entorno de desarrollo, diseñada para facilitar el uso de la electrónica en proyectos multidisciplinares[2].

A través de Arduino podemos recopilar multitud de información del entorno sin excesiva complejidad. Gracias a sus pines de entrada, nos permite jugar con una amplia gama de sensores (temperatura, luminosidad, presión, etc.) que nos brindan la capacidad de controlar o actuar sobre ciertos factores que le rodean, como por ejemplo: controlando luces, accionando motores, activando alarmas y muchos otros actuadores.

El entorno de desarrollo nos permite escribir, verificar y cargar en la memoria del microcontrolador de la placa Arduino programas mediante un cable USB conectado al ordenador. Dicho entorno dispone de su propio lenguaje de programación para el microcontrolador de la placa Arduino basado en Processing/Wiring.

Arduino se caracteriza por las siguientes ventajas:

- **Asequible:** Son relativamente baratos las placas comparados con otros micro controladores.
- **Multiplataforma**: El software de Arduino se puede ejecutar en diferentes sistemas operativos como Windows, Macintosh OSX y Linux.
- **Entorno de programación simple y claro:** Fácil de usar para principiantes y flexible para usuarios avanzados.
- **Software y Hardware Open Source:** Cualquiera que desee ampliar y mejorar tanto su diseño hardware de las placas como el entorno de desarrollo software y el propio lenguaje de programación. Además, su distribución es libre, es decir, puede utilizarse libremente para el desarrollo de cualquier tipo de proyecto sin haber adquirido ninguna licencia.

En el caso de este trabajo, la plataforma sobre la que se ha trabajado en la parte hardware ha sido el Arduino UNO R3. También ha sido necesario añadir a la placa un Arduino Wifi Shield SD para realizar la comunicación con otros dispositivos. A continuación, se detalla cada componente utilizado tanto hardware como software.

1.3.1.1 Arduino UNO R3

Es la última versión de la placa básica de Arduino. La placa tiene un tamaño de 75x53 mm. Su unidad de procesamiento consiste en un microcontrolador ATmega328. Puede ser alimentada mediante USB o alimentación externa y contiene pines tanto analógicos como digitales. La siguiente tabla resume sus componentes:

Microcontrolador	ATmega328
Voltaje operativo	5V
Voltaje de entrada(recomendado)	7-12V
Voltaje de entrada(limites)	6-20V
Pines digitales E/S	14 (de los cuales 6 proporcionan salida PWM)
Pines de entrada analógica	6
Corriente continua para pines E/S	40mA
Corriente continua para pines de 3.3V	50mA
Memoria Flash	32KB (ATmega328) de los cuales 0.5 KB son para el bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB(ATmega328)
Velocidades del reloj	16 MHz

En la Figura 1.1 muestra donde están ubicados los elementos más importantes que componen el módulo Arduino Uno.

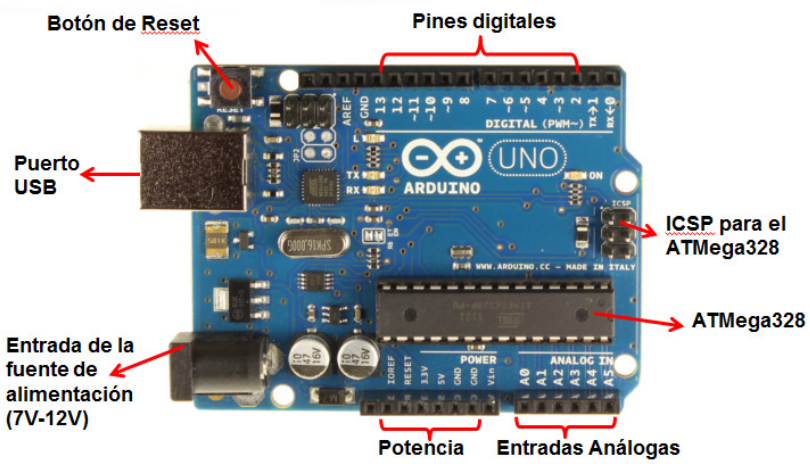


Figura 1.1: Arduino UNO R3 (elementos)

1.3.1.2 Arduino Wifi Shield SD

Esta Shield está pensado para los que le quieren añadir a la placa Arduino UNO la capacidad de conectarse inalámbricamente a una red TCP/IP. Tiene incorporado un chip ATmega 32UC3, con un microcontrolador de 32 bits que proporciona una pila IP completa (TCP y UDP).

También, como podemos observar (figura 1.2) que dispone de un conector propio mini USB, el cual no utilizamos para programar sino para actualizar el firmware del chip ATmega 32UC3. Dicho firmware será necesario actualizar para poder transmitir datos por la red [4]. Existen más características:

- Alimentación: 5V (proporcionado por la placa Arduino)
- Red: 802.11b/g
- Encriptaciones soportadas: WEP y WPA2-Personal
- Conexión con Arduino por el puerto SPI
- Zócalo para tarjeta Micro SD incorporado
- Pines ICSP
- Conexión FTDI para debug
- Conexión Mini-USB para actualizaciones de Firmware
- Propio botón de "reset"

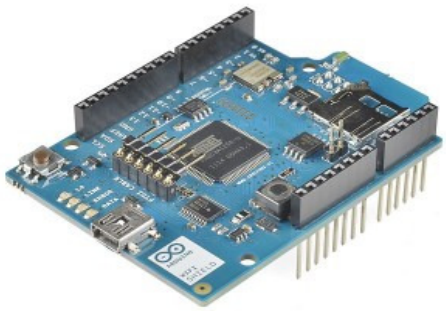


Figura 1.2: Arduino Wifi Shield SD

Por último, resaltar una serie de LEDs informativos que nos puede servir de utilidad para comprobar si la Shield está funcionando correctamente:

- L9 (led amarillo): conectado directamente al pin digital nº 9.
- LINK (led verde): que indica que se ha establecido conexión a una red.
- ERROR (led rojo): que indica si ha habido un error en la comunicación.

- DATA (led azul): que indica que hay datos transmitiéndose o recibiéndose en ese momento.

1.3.1.3 Sensores

La variedad de tipo de sensores que existen en el mercado para este trabajo se han elegido dos: temperatura y luminosidad.

Figura 1.3: Termistor NTC

- Termistor del tipo NTC (159-282-86001)[6] para el cálculo en grados Celcius del sensor.

Figura 1.4: Fotoresistencia LDR

- Fotoresistencia llamada LDR para calcular la intensidad de luz incidente[7].

1.3.1.4 Software

El entorno de desarrollo en Arduino (IDE) es el encargado de la gestión de la conexión entre el PC y el hardware de Arduino con el fin de establecer una comunicación entre ellos por medio de la carga de programas. Este entorno está implementado en Processing y la plataforma Arduino tiene un lenguaje propio. No obstante, hay que aclarar que el lenguaje Processing está construido internamente con código escrito en lenguaje Java, mientras que el lenguaje Arduino soporta funciones del estándar C y algunas de C++.

Para este trabajo se ha utilizado la última versión del IDE, que es la versión 1.6.5, sobre un sistema Windows. Además, este entorno proporciona librerías oficiales de los productos Arduino como la librería de programación "WiFi", entre otras, que nos permitirá realizar un conjunto de métodos para crear las conexiones a la red y transmisiones de datos por ella.

1.3.2 Android

Android es una plataforma móvil lanzado por la Open Handset Alliance, que consiste en una pila de software compuesto por un sistema operativo basado en Linux.

El desarrollo del software está ligado a la utilización de la máquina virtual Dalvik (Android Runtime) que permite el uso de Java como lenguaje de programación. La mayoría de las bibliotecas que son compatibles con JDK pueden ser desplegadas en un dispositivo Android.

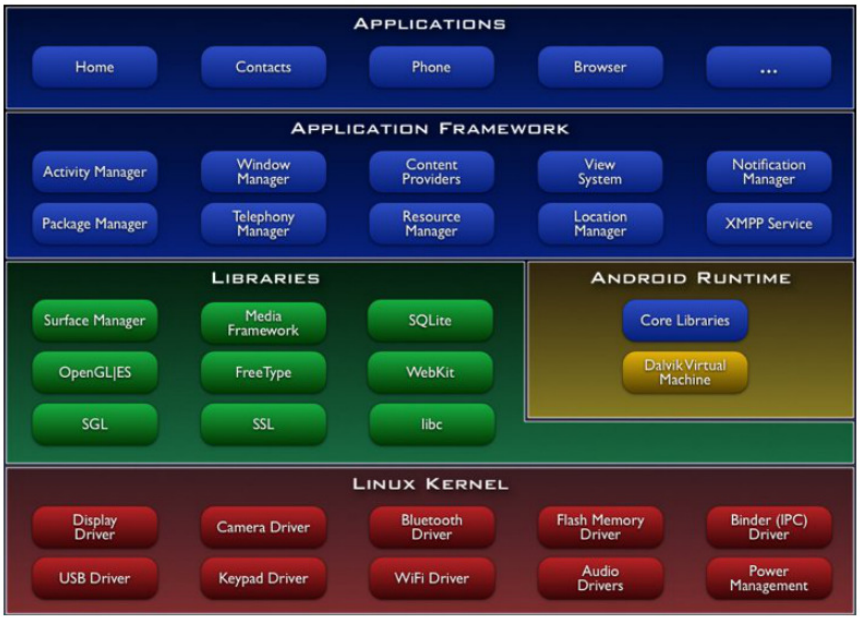


Figura 1.5: Arquitectura del sistema Android

La arquitectura de Android se compone de varias capas que se basan en un núcleo Linux como se muestra en la figura 1.5. La capa de aplicación incluye un conjunto de aplicaciones por defecto en el sistema operativo, como el calendario, contactos, etc. El marco de aplicación constará de los servicios predeterminados para la gestión de los recursos hardware (sensores, pantalla, etc.), software (alarmas, servicios de fondo) y la integración con recursos externos (sistemas de información de ubicación, notificación de servicios, etc.).

A pesar de que las aplicaciones de Android se distribuyen principalmente en el Android Market, estas se pueden distribuir libremente en Internet, una vez que se envasan en APK (Android Application Package). Sin embargo, si la aplicación no se compra en el Android Market, se corre el riesgo de adquirir malware para Android.

Desde la entrada de Google a esta plataforma se ha incrementado las capacidades de este sistema y su entorno de desarrollo. Para el desarrollo de la aplicación Android de nuestro trabajo utilizamos Android Developer Tools (ADT), que es un plugin para Eclipse que proporciona un conjunto de herramientas que se integran con el IDE de Eclipse.

En este trabajo introducimos Android mediante una aplicación con el principal propósito de monitorizar la información recogida de un dispositivo externo y su posterior almacenamiento.

1.3.3 Google Cloud Platform

Google Cloud Platform es una plataforma "Cloud Computing", traducido al español como la "Computación en la nube", de Google. Esta plataforma permite a los desarrolladores construir, probar y desplegar aplicaciones en la infraestructura altamente escalable y confiable de Google. Puede elegir la computación, almacenamiento y servicios de aplicaciones para sus soluciones web, móviles y de backend[10].

En Google Cloud Platform proporciona dos de los tipos de Cloud que existen, estos son Google App Engine como PaaS (Platform as a Service) y Compute Engine como IaaS (Infrastucture as a Service). En este trabajo se centrará más en los servicios de almacenamiento y herramientas de Google App Engine, los cuales explicaremos a continuación.

Figura 1.6: Google Cloud Platform

1.3.3.1 Google App Engine

Google App Engine es un servicio o herramienta para el alojamiento de aplicaciones web escalables y backends móviles sobre la infraestructura de Google de una forma gratuita hasta determinadas cuotas. El propósito es permitir al desarrollador crear fácilmente aplicaciones escalables sin ser un experto en sistemas[11].

App Engine tienes muchas características, entre ellas, es que sus aplicaciones soporta lenguajes de programación como Java, Python, Go y PHP. También funciona con herramientas de desarrollo como Eclipse, IntelliJ, Maven, Git, Jekins and PyCharm. En nuestra aplicación Web hemos utilizado el lenguaje Java versión 7, exactamente el estándar Java Servlet para aplicaciones Web (JSP, Servlets), en un entorno Eclipse 4.4 (Luna) gracias al Google Plugin para Eclipse[12]. Este plugin puede crear y desplegar aplicaciones en Google App Engine.

En el entorno de desarrollo de App Engine dispone de Software Development Kits (SDKs) que están disponibles en todos los lenguajes soportados. Este SDK contiene todas las librerías disponibles y APIs de App Engine. Simula todo los servicios y gestión tu aplicación en tu ordenador localmente y permite actualizarla en la Cloud, además de gestionar diferente versiones de ella. En nuestro caso utilizamos la última versión del App Engine SDK 1.9.19.

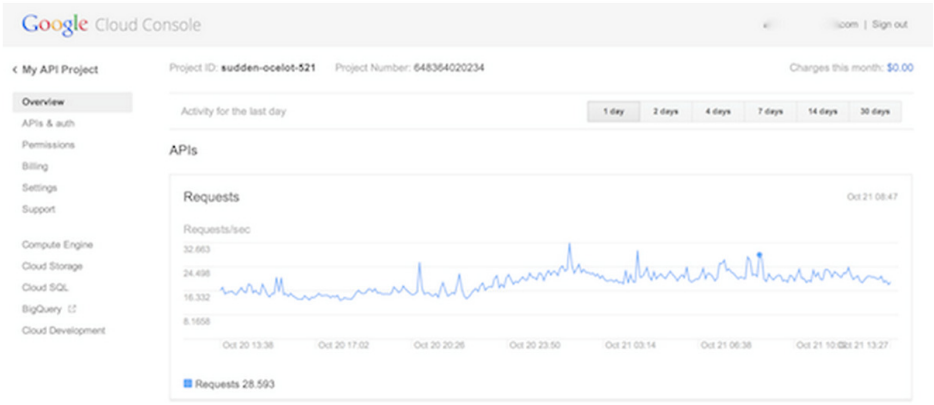


Figura 1.7: Google Cloud Console

Una herramienta que se caracteriza en App Engine es la Google Developers Console, que es una interfaz basada en web para la gestión y configuración de tus proyectos. En esta interfaz puede realizar, entre muchas otras:

- Obtener el ID del proyecto.
- Administrar tu equipo de trabajo
- Habilitar/Deshabilitar APIs y generar
- Visualizar registros de actividad
- Mostrar los recursos utilizados mediantes graficas de las diferente versiones
- Obtener los detalles de la cuota
- Consultar los metadatos de tu almacenamiento.

Además, Google proporciona un par de dominios para tu aplicación con la siguiente estructura "*midominio*.appspot.com". Otra característica es que pagas solo lo que usas, sin embargo en esto momento permiten cuotas gratuitas, como el límite de 1 gigabyte de almacenamiento y una cantidad de ancho de banda y CPU para cinco millones de visitas, entre otras, como vemos en la tabla[13].

Resource	Free Default Limit
Code & Static Data Storage	First 1 GB
Number of Indexes	200
Write Operations	50,000
Read Operations	50,000
Frontend Instances	28 free instance-hours per day
Logs data	1 GB
Incoming Bandwidth	1 GB

1.3.3.2 Cloud DataStore

Cloud DataStore es una base de datos NoSQL altamente escalable para aplicaciones web o móviles. En App Engine para Java permite el uso de dos estándares de API diferentes para el almacenamiento de datos: Java Data Object (JDO) y Java Persistence API (JPA). Estas interfaces las proporciona DataNucleus Access Platform, una implementación de software libre de varios estándares de persistencia Java, con un adaptador para Google DataStore.

Los objetos de datos en el DataStore se conocen como entidades (entities). Una entidad tiene una o más propiedades con nombre, cada uno de los cuales puede tener uno o más valores. DataStore es compatible con una variedad de tipos de datos para los valores de las propiedades.

Cada entidad (entity) en el DataStore tiene una clave (key) que identifica de manera única. La clave consiste en los siguientes componentes:

- El espacio de nombres de la entidad (namespace).
- El tipo de la entidad (kind), que se caracteriza por las consultas en el DataStore
- Un identificador (identifier) para la entidad individual, que puede ser tanto una cadena de clave de nombre como un identificador numérico entero.
- Y opcionalmente, una ruta ancestro (ancestor path) para localizar la entidad dentro de la jerarquía de DataStore.

Las consultas de DataStore recupera entidades del App Engine, que resolverá un conjunto específico de condiciones. Las consultas funcionan con entidades de un tipo determinado, puede especificar filtros en las propiedades de valores, claves y ancestros de las entidades, y pueden devolver cero o más entidades como resultado. Una consulta también puede especificar criterios de ordenación para secuenciar los resultados por parte de las propiedades de valores.

A diferencia de las bases de datos relacionales tradicionales, el tipo de consultas puede ser más restrictivas debido por los índices pre construidos. Por ello, en DataStore no podemos realizar las siguientes consultas:

- Operaciones con Join
- Desigualdad filtrado en varias propiedades
- Filtrado de datos basados en resultados de una subconsulta.

Durante el desarrollo de la aplicación alojada en Google App Engine se diseña una base de datos con JDO en la que se aplicará estas propiedades y se realizarán varios tipos de consultas.

1.3.3.3 Google Cloud Endpoints

Google Cloud Endpoints es un servicio que te permite definir la lógica de negocios en App Engine a través de la creación de servicios REST (Representational State Transfer). También, se define como un conjunto de herramientas que nos permiten, de forma sencilla e incluso automáticamente, generar las APIs para poder comunicar aplicaciones clientes (webs y móviles) con un backend, como Android, iOS y clientes JavaScript.

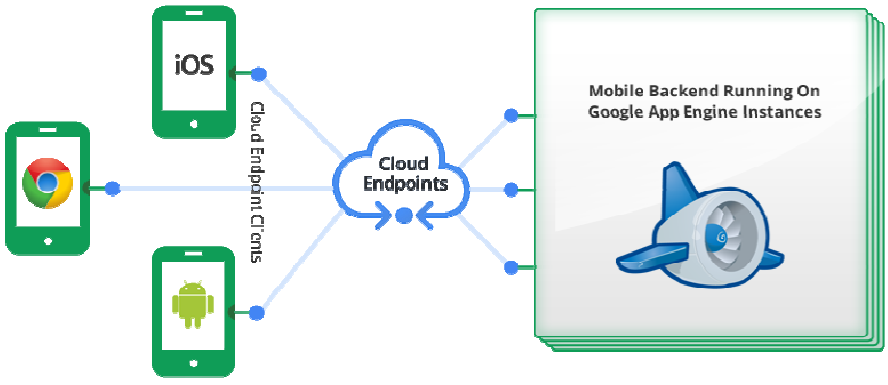


Figura 1.8: Arquitectura Básica Endpoints

Con Google Cloud Endpoints vamos a disponer de unos mecanismos sencillos para poder crear, exponer y consumir nuestra API REST con la que compartiremos datos entre nuestras aplicaciones clientes y la parte backend. Toda la parte backend se va almacenar y gestionar en el Google App Engine, por lo que los desarrolladores pueden usar servicios como DataStore.

Podemos decir, que lo que vamos a tener, van a ser una serie de objetos de datos a los que vamos a poder realizar las típicas operaciones CRUD. Vamos a disponer de métodos para consultas, dar de alta nuevos registros, actualizar y eliminar registros. Y todo ello, se gestionará a través de una API REST.

Como hemos comentado anteriormente utilizamos el Google Plugin para Eclipse que te permite crear y utilizar fácilmente Cloud Endpoints. Además, este plugin proporciona características que te ayudan a desarrollar el Cloud Endpoints como la generación automática de la configuración de la API y de las clases Endpoints, los cuales contienen código para crear, obtener, listar, actualizar y borrar las clases de las entidades del DataStore. Estas clases son simplemente un POJO con anotaciones de persistencias de JPA/JDO.

Para usar Endpoints en Java y en Android, como es nuestro caso, debe estar seguro de usar la última versión de Google App Engine Java SDK y Android Development Tools (ADT). Para clientes Android, el Google Plugin para Eclipse proporciona la generación de un proyecto App Engine Backend y de las librerías cliente para Android, estas librerías son copiadas sobre las librerías del cliente generadas y sus dependencias al proyecto Android.

2. Especificación y Diseño

En esta fase del trabajo vamos a explicar la especificación del sistema a desarrollar y su diseño, tanto de la aplicación móvil en Android como de la aplicación Web alojada en la Cloud.

2.1 Requisitos

Los requisitos muestran qué elementos y funciones son necesarias para un proyecto en particular. Este tipo de requisitos se llama requisitos funcionales. En nuestro trabajo describirán las funcionalidades o los servicios que se espera que el sistema suministre al usuario.

Consideramos los requisitos que tendrán que cumplirse en las aplicación Android y Web, aunque existirá algunos que solo se cumplen en la aplicación móvil y otros en la aplicación Web. En nuestra aplicación hay dos roles, que son el usuario y el administrador, que cumplirán algunos requisitos diferentes.

Otros requisitos son los no funcionales que nos permitirán indicar las restricciones que debe cumplir el sistema.

2.1.1 Requisitos Funcionales

1. **Acceder al sistema**: El usuario debe identificarse mediante su email y su contraseña para entrar al sistema.
2. **Salir del sistema**: El usuario puede salir del sistema desde un botón dentro de la aplicación o desde un botón externo a la aplicación.
3. **Monitorizar los datos** *(Solo Aplicación Android)*: El usuario puede activar la monitorización de los valores manualmente (mediante un botón) en la aplicación móvil.
4. **Guardar los datos monitorizados** *(Solo Aplicación Android)*: El usuario puede guardar los valores monitorizados en la aplicación móvil (mediante un botón) en la base de datos del sistema.
5. **Mostrar historial de un tipo de dato**: El usuario puede ver el historial de los valores de temperatura o luminosidad anteriormente guardado en la base de datos del sistema (mediante un botón del panel principal de la aplicación).
6. **Mostrar historial completo de los datos**: El administrador podrá ver el historial de todos los datos almacenados en la base de datos del sistema.
7. **Filtrar datos del historial**: El usuario puede filtrar el historial de un tipo dato por valores o por su fecha de almacenamiento.
8. **Registrar usuario** *(Solo Aplicación Web)*: El sistema debe permitir registrarse a un usuario sin entrar en la aplicación.

9. **Editar su información de usuario** *(Solo Aplicación Web)*: El usuario puede editar sus datos como email, nombre y contraseña. El rol de un usuario solo podrá modificarlo un administrador del sistema.
10. **Listar usuarios***:* Solamente el administrador puede visualizar todos los usuarios y administradores registrados en el sistema.
11. **Eliminar usuario** *(Solo Aplicación Web):* El administrador puede eliminar un usuario del sistema.
12. **Eliminar Datos**: El usuario puede eliminar datos asignados de su historial almacenado en la bases de datos del sistema. El administrador tendrá la opción de borrar datos del sistema.
13. **Administrar alertas** *(Solo Aplicación Web):* El usuario podrá consultar, crear, editar, eliminar y activar alertas del sistema.

2.1.2 Requisitos No Funcionales:

1. El sistema debe obtener y monitorizar los valores automáticamente en intervalo de tiempo lo suficientemente corto para que la aplicación tenga una buena precisión.
2. El sistema debe procesar las consultas al servidor lo suficientemente rápido para que la aplicación tenga fluidez.
3. El sistema debe utilizar los recursos hardware disponible de la manera más efectiva posible.
4. El sistema debe facilitar al máximo la posibilidad de añadir nuevas funcionalidades.
5. El sistema debe utilizar hardware libre y software libre
6. El sistema debe ser fiable, minimizando la aparición de fallos.
7. El sistema debe ser fácil de utilizar por el usuario de manera que tenga una interfaz intuitiva y sencilla.

2.2 Casos de uso

A continuación detallaremos los principales casos de uso desarrollado en este proyecto. En la siguiente figura 2.1 mostramos el modelo de casos de uso realizado en UML.

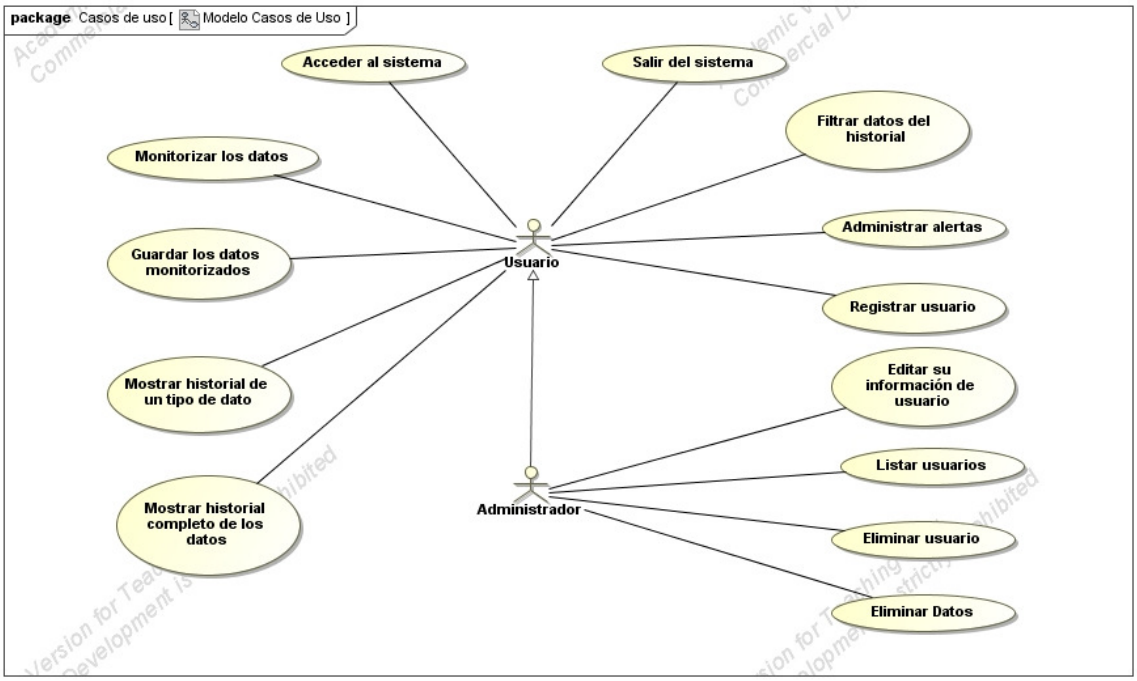


Figura 2.1: Modelo de Casos de Uso

1. Acceder al sistema.

1.1. Contexto de uso: Cuando el usuario quiere acceder a la aplicación introduciendo su email y contraseña.

1.2. Nivel: Usuario

1.3. Participantes y objetivos:

Usuario: Acceder a la aplicación correctamente con su email y contraseña.

1.4. Precondiciones:

1.4.1. Debe estar instalada la aplicación en el móvil

1.4.2. Debe estar abierta la aplicación en el móvil

1.4.3. Debe haber accedido a la dirección del servidor en un navegador web.

1.5. Garantías mínimas: Ninguna

1.6. Garantías de éxito: Se accede a la aplicación y aparece el menú principal.

1.7. Escenario de éxito principal:

1. El usuario introduce su email y contraseña en la aplicación.
2. El sistema comprueba la existencia en la base de datos del email y contraseña.

1.8. Extensiones:

2.a. Si el email o la contraseña no existen en la base de datos, entonces se le informa al usuario con un mensaje.

2.b. Si existen, entonces se carga el menú principal de la aplicación.

2. **Salir del sistema.**

2.1. Contexto de uso: Cuando el usuario quiere salir de la aplicación por algún motivo.

2.2. Nivel: Usuario

2.3. Participantes y objetivos:

Usuario: Salir de la aplicación

2.4. Precondiciones:

2.4.1. Debe estar logueado en la aplicación móvil.

2.4.2. Debe estar logueado en la aplicación web.

2.5. Garantías mínimas: Ninguna

2.6. Garantías de éxito: Salir del sistema sin ningún problema.

2.7. Escenario de éxito principal:

1. El usuario pulsa el botón de cerrar sesión o logout.

2.8. Extensiones: Ninguna

3. **Monitorizar los datos** (Solo para Android).

3.1. Contexto de uso: Cuando la monitorización esta parada y el usuario quiere volver a activar la monitorización, dicho usuario tendrá que pulsar un botón manualmente.

3.2. Nivel: Usuario.

3.3. Participantes y objetivos:

Usuario: Activar la monitorización de los valores de temperatura y luminosidad.

3.4. Precondiciones:

3.4.1. Debe estar logueado en la aplicación móvil.

3.4.2. El sistema debe tener parada la monitorización de valores de temperatura y luminosidad.

3.5. Garantías mínimas: Ninguna

3.6. Garantías de éxito: Se vuelve a monitorizar los valores en la pantalla principal.

3.7. Escenario de éxito principal:

1. El usuario pulsa el botón de activar monitorización
2. El sistema recibe la acción y se conecta al Arduino para monitorizar.

3.8. Extensiones:

2.a. Si el sistema no se puede conectar al Arduino, entonces se le informa al usuario con un mensaje.

4. **Guardar los datos monitorizados** (Solo para Android).

4.1. Contexto de uso: Cuando el usuario quiere guardar manualmente los valores monitorizados hasta el momento, dicho usuario pulsa un botón para hacerlo.

4.2. Nivel: Usuario

4.3. Participantes y objetivos:

Usuario: Guardar valores en la base de datos.

4.4. Precondiciones:

4.4.1. Debe estar logueado en la aplicación móvil.

4.4.2. El sistema debe estar monitorizando los valores de temperatura y luminosidad.

4.5. Garantías mínimas: Ninguna

4.6. Garantías de éxito: Los valores son guardados en la base de datos.

4.7. Escenario de éxito principal:

1. El usuario pulsa el botón de Guardar datos en la aplicación Android.
2. El sistema recoge todos los datos monitorizados y los guarda en la base de datos.

4.8. Extensiones: Ninguna

5. **Mostrar historial de un tipo de dato**.

5.1. Contexto de uso: Cuando el usuario desea ver los valores guardados de temperatura o luminosidad, dicho usuario tendrá que pulsar un botón del panel principal.

5.2. Nivel: Usuario.

5.3. Participantes y objetivos:

Usuario: Visualizar su historial de los valores guardados en la base de datos.

5.4. Precondiciones:

5.4.1. Debe estar logueado en la aplicación móvil.

5.4.2. Debe estar logueado en la aplicación web.

5.5. Garantías mínimas: Aparece una pantalla vacía.

5.6. Garantías de éxito: Muestra otra pantalla con el historial de valores.

5.7. Escenario de éxito principal:

1. El usuario pulsa el botón correspondiente de historial del tipo de dato a visualizar.
2. El sistema muestra los datos del tipo seleccionado.

5.8. Extensiones: Ninguna

6. **Mostrar historial completo de los datos** (Solo Administrador):

6.1. Contexto de uso: Cuando el administrador quiere ver todos los datos almacenados de la base de datos del sistema, tanto valores de temperatura como luminosidad.

6.2. Nivel: Administrador

6.3. Participantes y objetivos:

Administrador: Mostrar todos los datos almacenados del sistema.

6.4. Precondiciones:

6.4.1. Debe estar logueado en la aplicación móvil.

6.4.2. Debe estar logueado en la aplicación web.

6.5. Garantías mínimas: Muestra una pantalla vacia.

6.6. Garantías de éxito: Se muestra el historial de todos los datos almacenados del sistema.

6.7. Escenario de éxito principal:

1. El administrador pulsa el botón para mostrar todos los datos.
2. El sistema muestra todos los datos almacenados del sistema.

6.8. Extensiones: Ninguna

7. **Filtrar datos del historial**:

7.1. Contexto de uso: Cuando el usuario quiere realizar un filtro en los listados de datos entre dos valores o fechas.

7.2. Nivel: Usuario

7.3. Participantes y objetivos:

Usuario: Filtra los datos mostrados en el historial.

7.4. Precondiciones:

7.4.1. Debe estar logueado en la aplicación móvil.

7.4.2. Debe estar logueado en la aplicación web.

7.4.3. Debe estar mostrando un historial de los datos.

7.5. Garantías mínimas: Ninguna

7.6. Garantías de éxito: Se muestra los datos correctamente según el filtrado.

7.7. Escenario de éxito principal:

1. El usuario introduce los valores o fechas para filtrar en el historial.
2. El sistema comprueba datos introducidos y muestra el listado filtrado de datos.

7.8. Extensiones: Ninguna

8. **Registrar usuario** (Solo Aplicación web):

8.1. Contexto de uso: Cuando el administrador quiere insertar un nuevo usuario en el sistema.

8.2. Nivel: Administrador.

8.3. Participantes y objetivos:

Administrador: Insertar un nuevo usuario.

8.4. Precondiciones:

8.4.1. Debe estar logueado en la aplicación móvil.

8.4.2. Debe estar logueado en la aplicación web.

8.5. Garantías mínimas: Ninguna

8.6. Garantías de éxito: El usuario será correctamente insertado en el sistema.

8.7. Escenario de éxito principal:

1. El administrador introduce el email, nombre y "contraseña".
2. El sistema recoge y comprueba los datos introducidos.

8.8. Extensiones:

2.a. Si el email existe, el sistema informará al administrador con un mensaje de email existente.

2.b. Si el email no existe, el sistema insertará el usuario e informará al administrador.

9. **Editar su información de usuario**:

9.1. Contexto de uso: Cuando el usuario quiere actualizar sus datos de usuario.

9.2. Nivel: Usuario.

9.3. Participantes y objetivos:

Usuario: Actualizar sus datos de usuario.

9.4. Precondiciones:

9.4.1. Debe estar logueado en la aplicación móvil.

9.4.2. Debe estar logueado en la aplicación web.

9.5. Garantías mínimas: Ninguna

9.6. Garantías de éxito: Se actualizan correctamente los datos modificados.

9.7. Escenario de éxito principal:

1. El usuario modifica sus datos (email, nombre o contraseña)
2. El sistema recoge los datos y comprueba el email introducido.

9.8. Extensiones:

2.a. Si el email modificado existe, el sistema informará al usuario con un mensaje de email existente.

2.b. Si el email es el mismo del usuario, el sistema actualizará los demás datos introducidos.

2.c. Si el email no existe, el sistema actualizará todos los datos del usuario e informará al usuario.

10. **Listar usuarios** (Solo Administrador):

10.1. Contexto de uso: Cuando el administrador quiere ver todos los usuarios del sistema.

10.2. Nivel: Administrador.

10.3. Participantes y objetivos:

Administrador: Mostrar los usuarios del sistema.

10.4. Precondiciones:

10.4.1. Debe estar logueado en la aplicación móvil.

10.4.2. Debe estar logueado en la aplicación web.

10.5. Garantías mínimas: Muestra una pantalla vacía.

10.6. Garantías de éxito: Se muestra el listado de usuario del sistema.

10.7. Escenario de éxito principal:

1. El administrador pulsa el botón de mostrar usuarios.
2. El sistema muestra el listado de los usuarios del sistema.

10.8. Extensiones: Ninguna

11. **Eliminar usuario** (Solo Administrador):

11.1. Contexto de uso: Cuando el administrador quiere eliminar un usuario del sistema.

11.2. Nivel: Administrador

11.3. Participantes y objetivos:

Administrador: Eliminar el usuario del sistema.

11.4. Precondiciones:

11.4.1. Debe estar logueado en la aplicación móvil.

11.4.2. Debe estar logueado en la aplicación web.

11.5. Garantías mínimas: Ninguna

11.6. Garantías de éxito: Se elimina el usuario seleccionado del sistema por el administrador.

11.7. Escenario de éxito principal:

1. El administrador selecciona un usuario del sistema y pulsa el botón de eliminar.
2. El sistema elimina usuario seleccionado del sistema e informa al administrador con un mensaje.

11.8. Extensiones: Ninguna

12. **Eliminar datos** (Solo Administrador):

12.1. Contexto de uso: Cuando el administrador quiere eliminar un dato del sistema.

12.2. Nivel: Administrador

12.3. Participantes y objetivos:

Administrador: Eliminar un dato del sistema.

12.4. Precondiciones:

12.4.1. Debe estar logueado en la aplicación móvil.

12.4.2. Debe estar logueado en la aplicación web.

12.5. Garantías mínimas: Ninguna

12.6. Garantías de éxito: Se elimina el dato seleccionado del sistema por el administrador.

12.7. Escenario de éxito principal:

1. El administrador selecciona un dato del sistema y pulsa el botón de eliminar.
2. El sistema elimina el dato seleccionado e informa al administrador con un mensaje.

12.8. Extensiones: Ninguna

13. **Administrar alertas** (Solo Aplicación Web):

13.1. Contexto de uso: Cuando el usuario quiere realizar alguna acción como crear, editar, consultar y eliminar una alerta, también activar o desactivar la alerta.

13.2. Nivel: Usuario.

13.3. Participantes y objetivos:

Usuario: Administrar las alertas del sistema.

13.4. Precondiciones:

13.4.1. Debe estar logueado en la aplicación móvil.

13.4.2. Debe estar logueado en la aplicación web.

13.5. Garantías mínimas: Ninguna

13.6. Garantías de éxito: Se crea, edita, consulta, o elimina una alerta del sistema. O se activa/desactiva.

13.7. Escenario de éxito principal:

1. El usuario realiza una acción sobre una alerta del sistema.
2. El sistema ejecuta la acción seleccionada de la alerta específica e informa al usuario con un mensaje.

13.8. Extensiones: Ninguna

2.3 Arquitectura

En este apartado, mostramos la representación de alto nivel de la arquitectura que nos permite visualizar los distintos componentes que integran nuestro sistema, además de las comunicaciones entre los componentes, como vemos en la figura 2.2.

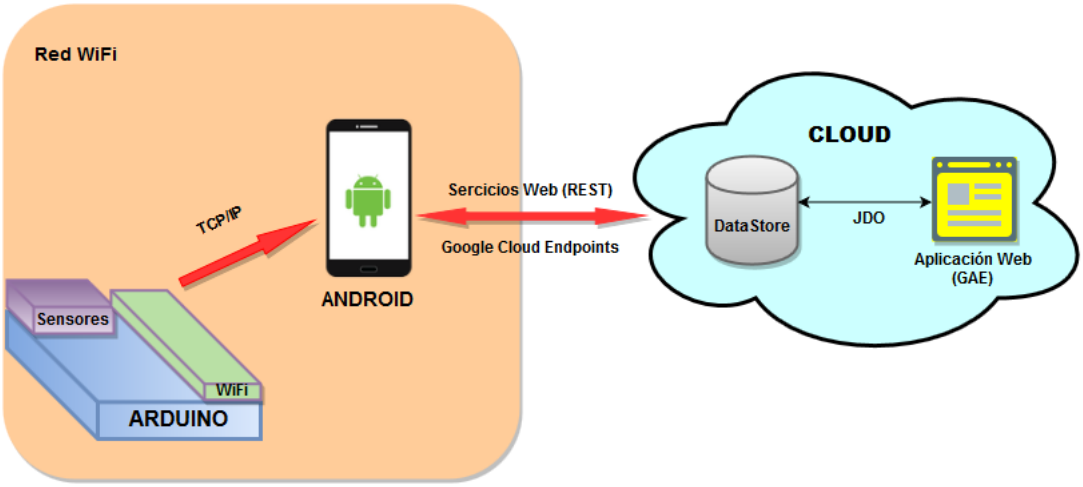


Figura 2.2: Arquitectura del Sistema

Como podemos observar en la arquitectura, hay tres importantes bloques o componentes, en los que recae el funcionamiento de este sistema: el módulo Arduino, el dispositivo móvil Android y la Cloud Computing ("la nube"). En este sistema vemos que tiene un papel esencial son dos procesos para comunicar dichos componentes entre ellos.

El proceso de monitorización empieza en el módulo Arduino, el cual realiza la función de recoger información de unos sensores (temperatura y luminosidad) conectado con el módulo. Posteriormente, ésta información se procesa y se envía al dispositivo móvil a través de la red Wifi donde están los dispositivos. La información se recibe y se muestra en la aplicación Android por pantalla al usuario, y así continuamente.

El otro proceso apreciable es la comunicación entre el dispositivo móvil y la nube. En este proceso existen dos principales acciones: una de almacenar los datos en la nube y otra de cargar los datos de la nube. La primera acción se consigue utilizando, desde la aplicación Android, unos servicios web que dispone la aplicación Web alojada en la nube que te permite añadir recursos a la base de datos. La acción de cargar los datos se logra solicitando otros servicios web de la aplicación web para traer los datos de la base de datos.

También podemos apreciar que dentro de la Cloud tenemos alojada una aplicación Web. Esta aplicación nos facilita efectuar acciones sobre la base de datos a través de un cliente Web, con ayuda de una interfaz (JDO) estándar para el almacenamiento de objetos en una base de datos.

Dentro de este apartado podemos mencionar el concepto de patrón de diseño ya son la base para la búsquedas de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces.

El primer patrón a destacar es el utilizado para desarrollar las aplicaciones Android y Web, este es el patrón arquitectónico Modelo-Vista-Controlador (MVC), el cual nos permite separar el modelo de negocio, la interfaz de usuario y la lógica de negocio.

Otro patrón en la arquitectura que se ha utilizado en el sistema es el Cliente-Servidor para las comunicaciones entre componentes. Esté nos permite tener un control centralizado que refuerza la seguridad de las comunicaciones y facilita su mantenimiento. Para la comunicación entre Arduino y Android lo aplicamos para efectuar el protocolo TCP/IP. Para la comunicación entre Android y la Cloud se aplica gracias los servicios (Endpoints) generado en el servidor.

Los patrones de diseño también pueden estar enfocados a la implementación. Para estos tipos se han utilizado el patrón *Singleton* para la creación de los cuadros de dialogo en Android, y el patrón estructural *Fachada*, o *Facade*, que nos facilita un interfaz de alto nivel para la capa de persistencia que actúa sobre la base de datos.

2.4 Diseño

En este capítulo se detalla el diseño de los componentes que forman parte de la arquitectura del sistema. En el siguiente diagramas de paquetes (*Figura 2.3*) reflejamos en el sistema las estructuras de cada modulo.

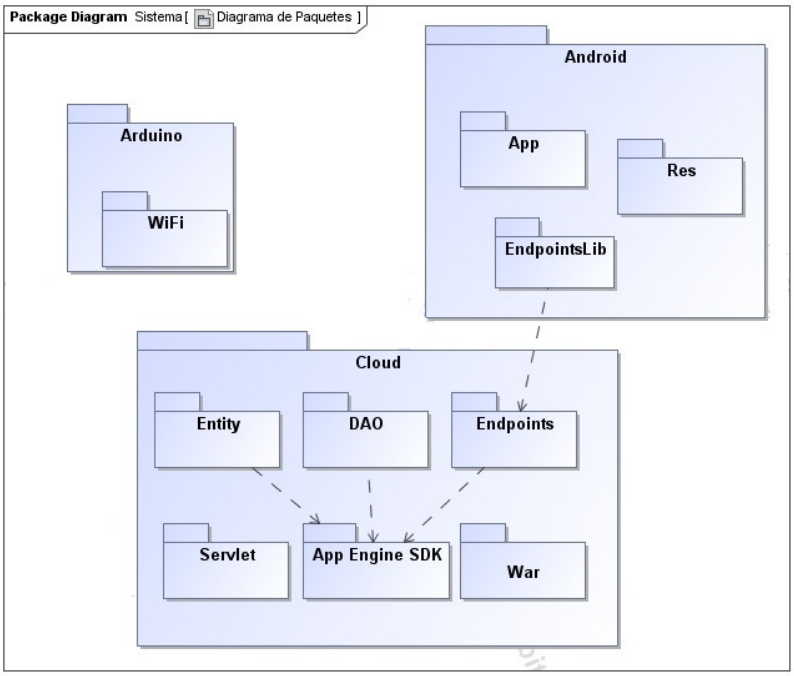


Figura 2.3: Diagrama de paquetes

En este diagrama vemos que cada modulo del sistema está representado como un paquete. Dentro del paquete Arduino hemos incluido la librería "WiFi" que proporciona el IDE como paquete ya que se utiliza en este modulo.

El modulo Cloud estará formado cinco paquetes: Entity, DAO, Servlet, Endpoints y War. En este último estarán los archivos de configuraciones y paginas de la aplicación Web. Además hemos añadimos la librería App Engine SDK ya que se usa en la mayoría de los paquetes mencionado.

En el modulo Android hay un paquete llamada "App" que contendrá las clases, o "Activity" de la aplicación. Igualmente habrá un paquete llamado res para almacenar los xml de las vistas, imágenes o cadenas utilizadas en la aplicación. Además, se ha intentado reflejar las librerías cliente generada por los Endpoints de la Cloud como un solo paquete, aunque en realidad será una librería para cada Endpoint creado.

2.4.1 Modelo

El siguiente diagrama de clases (*Figura 2.4*) representa el modelo de la base de datos alojada en la Cloud, la cual posee un importante papel para desarrollar el propósito general en este sistema. Por ello nos referiremos a una clase como entidad.

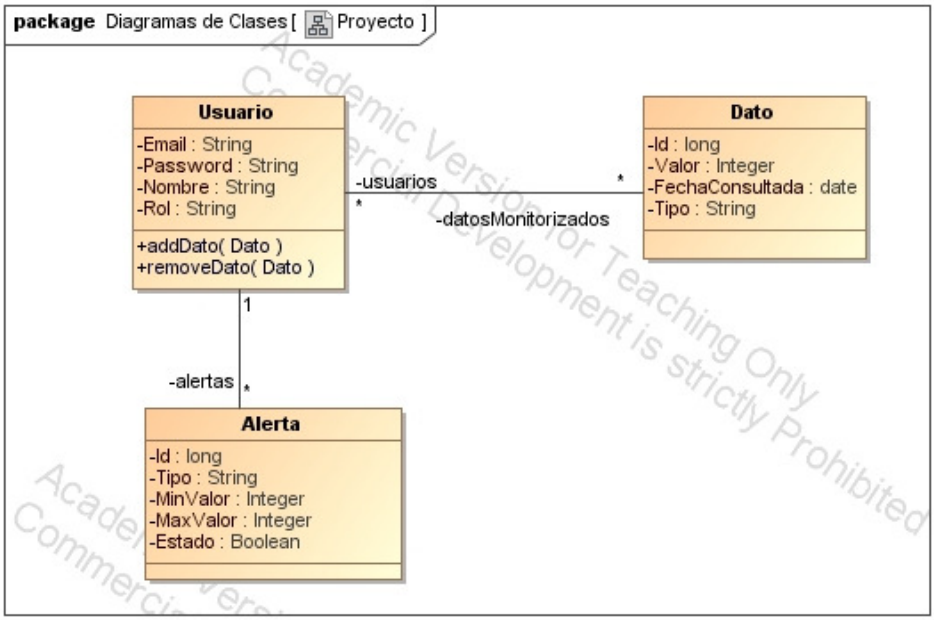


Figura 2.4: Diagrama de clases del modelo

Como podemos ver, el modelo consta de tres entidades sencillas: Usuario, Dato y Alerta. Además tenemos dos relaciones, una de ellas son entre la entidad Usuario y Dato donde el usuario puede contener varios datos monitorizados almacenado hasta el momento. La otra relación es Usuario-Alerta, la cual nos permite configurar alertas en el sistema.

El usuario tiene un *email* y un *password* para acceder al sistema, donde el email será la clave primaria de esta entidad. Igualmente podemos guardar el *nombre* del usuario. Además, tenemos un atributo para diferenciar el tipo de usuario que se conecta al sistema llamado *rol*, que contendrá "Usuario" o "Administrador". Por otro lado, hay dos métodos o operaciones que facilitarán añadir y eliminar datos asignado al usuario, llamados *addDato(Dato)* y *removeDato(Dato).*

El dato tendrá un identificador (*id*) numérico de tipo Long, el *valor* del dato monitorizado de tipo integer y el *tipo* de dato guardado, que en nuestro caso será "Temperatura" o "Luminosidad". También guardaremos la *fecha* de la monitorización de dicho dato.

La alerta tendrá un identificador (*id*) numérico de tipo Long, un *valor mínimo* y un *valor máximo* que configura el usuario para parar el sistema, asimismo el *tipo* de dato. Dicha alerta puede ser activada o desactivada con el *atributo* estado de tipo booleano.

2.4.2 Módulo Arduino

Este modulo engloba la parte hardware del sistema, igualmente contiene componentes electrónicos como son resistencias o leds. Los elementos físicos más relevantes del sistema son la placa Arduino, la Wifi Shield y los sensores, ya que con estos podremos conseguir el objetivo de comunicar la plataforma Arduino y el dispositivo móvil.

Para llevar a cabo este módulo del sistema es necesario diseñar un circuito electrónico que conecte los sensores con la placa Arduino como vemos en la *Figura 2.5.* Podemos observar que en el esquema las conexiones a los pines se realizan sobre la placa Arduino, aunque en la realidad dichas conexiones serán en la Wifi Shield, ya que está se acoplará en el modulo Arduino a través de los pines SPI del Arduino. Por lo que el microcontrolador obtendrá la información de los pines conectados.

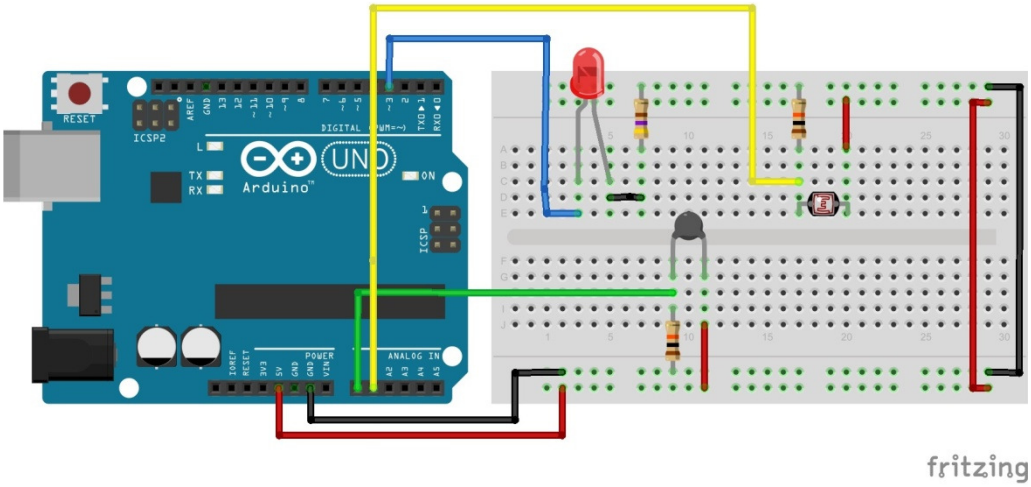


Figura 2.5: Esquema del circuito

Para el diseño de este circuito hemos utilizados, aparte del modulo Arduino, una protoboard donde conectaremos los siguientes componentes electrónicos:

- Una resistencia de 470 Ω
- Dos resistencias de 10K Ω
- Un led rojo
- Un sensor de temperatura NTC-TMP
- Un LDR

2.4.2.1 Programa Software

En este modulo será necesario ejecutar un programa software cargado en el microcontrolador, conocido como sketch, para obtener y transmitir la información de los diferentes sensores conectados al Arduino. Para ello, explicaremos el proceso iterativo en el siguiente diagrama de flujos del programa.

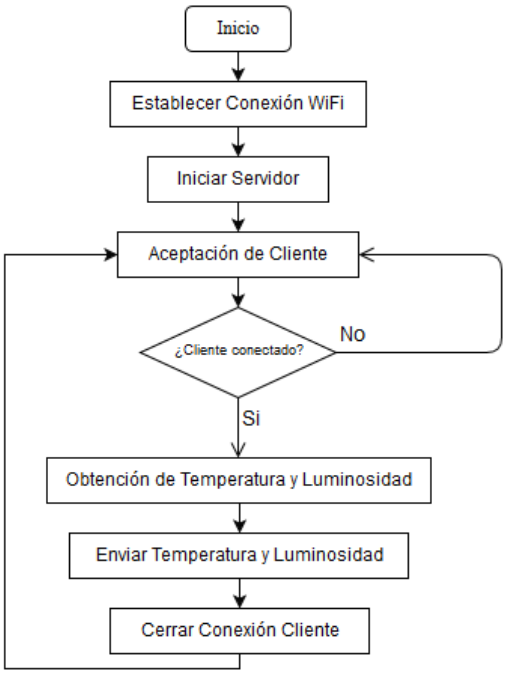


Figura 2.6: Diagrama de flujo del programa Arduino

Como podemos observar empezamos realizando un proceso de configuración, donde establecemos la conexión con la red Wifi y dar comienzo la escucha del servidor. Una vez configurado, se da lugar a un bucle infinito, donde primero espera aceptar un cliente de la red y luego procede a obtener un par de valores por las entradas analógicas conectadas a los sensores de temperatura y luminosidad. Cada valor se transformará y normalizará, mediante métodos diferente, en grados Celsius la temperatura ambiente y en % de luz del entorno.

Por último, los valores transformados serán transmitidos desde servidor Arduino y escribirá sobre el pin digital donde tendremos conectado el led, que se encenderá y se apagará cada cierto tiempo para señalizar que se están transmitiendo los datos. Para finalizar, se cierra la conexión con el cliente para volver a comenzar el proceso.

Este proceso de transmisión que hemos descrito se llevará a cabo por el protocolo TCP/IP, el cual está basado en la programación de sockets. Este protocolo te permite comunicar diferentes dispositivos dentro de una misma red entre sí y garantiza que la información llega correctamente a su destinatario. Por lo que nos asegura que no se perderá ningún dato en la transmisión. En nuestro caso solo enviaremos datos desde el Servidor al Cliente Android.

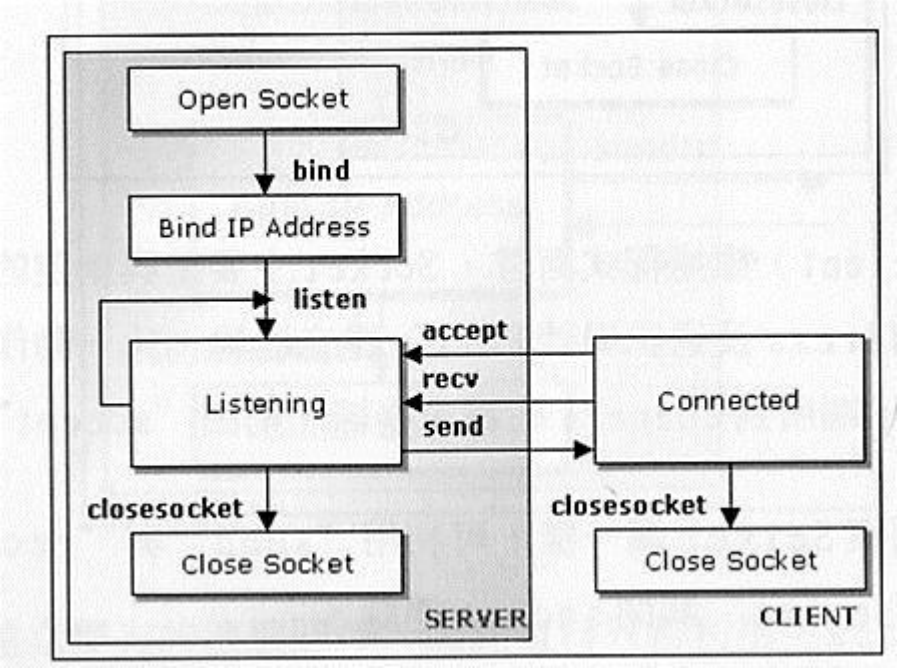


Figura 2.7: Protocolo TCP/IP basado en socket

Como ya hemos mencionado en otros apartados, el software de Arduino nos proporciona librerías que podemos utilizar gracias al IDE para facilitar la interacción entre el modulo y el PC. Una de ella, es la librería WiFi que nos permitirá crear socket para comunicarnos a través de la red. A continuación explicaremos las clases y métodos utilizados en este modulo:

- **WiFi:** Esta clase te permite inicializar la librería Ethernet y configuraciones de red.
    - **status():** Te devuelve el estado de la conexión, como por ejemplo el estado WL_CONNECTED que se le asigna cuando se conecta a una red WiFi o el estado WL_NO_SHIELD que informa de si está conectada la WiFi Shield.
    - **begin(ssid,pass):** Inicializa la conexión con la red WiFi mediante el nombre de la red (SSID) a conectar y el password que usa la red WPA encriptada para su seguridad.
    - **localIP():** Obtiene la dirección IP asignada a la WiFi Shield, devuelve un tipo IPAdress.
- **WiFiServer:** Esta clase te permite crear servicios con los que puedes enviar y recibir datos de un cliente conectado.
    - **server(port):** Crea un servidor que te permite escuchar conexiones a través de un puerto especifico.
    - **begin():** Método para el servidor empiece a escuchar.

- **available():** Con este método obtienes clientes que están conectado al servidor y preparados para transmitir o recibir.
- **println(data):** Permite escribir datos sobre el buffer para enviárselo a los clientes.

- **WiFiClient:** Esta clase te permite crear clientes que pueden conectarse a servidores y enviar/recibir dato.
  - **connected():** Es una función que te devuelve true si está conectado el cliente, y false si no está conectado.
  - **flush():** Desecha los bytes que ha sido escrito al cliente pero no aun leído.
  - **stop():** Desconecta la conexión con el servidor.

Otras librerías y funciones a tener en cuentas de Arduino para poder escribir en el monitor serie del IDE o por la entrada/salida de la placa son:

- **SPI (Serial Peripheral Interface):** Esta librería te proporciona poder conectar tu Arduino a la shield usando SPI mediante la librería WiFi.
- **Serial:** Esta clase se utiliza principalmente para comunicar la placa Arduino con otros dispositivos, por ejemplo, con el PC. Para ello utiliza los puertos serial del Arduino.
  - **begin(speed)**: Configura los baudios para la transmisión de datos en serie. Utilizaremos para mostrar el proceso por el monitor serie del IDE.
  - **print() / println():** Imprime datos al puerto serie como texto ASCII legible.
- **pinMode(pin,mode):** Configura el pin especificado como entrada (INPUT) o salida (OUTPUT). En nuestro caso será de salida para el led.
- **analogRead(pin):** Este método te permite leer el valor de un pin analógico especifico. Dependiendo del voltaje de 0-5V devolverá un valor entre 0-1023
- **digitalWrite(pin,value):** Este método te permite volcar por el pin especificado un voltaje de 5V con el valor HIGH o un voltaje de 0V con el valor LOW. Lo utilizaremos para

2.4.3 Aplicación Android

La aplicación Android creada en este trabajo tiene una función fundamental en el sistema a desarrollar, como hemos visto en la arquitectura (Figura 2.2), que es el intermediario entre las plataformas Arduino y Cloud. Esta aplicación es la encargada de realizar el proceso de monitorización de los sensores de temperatura y luminosidad, además del proceso de almacenamiento de la información extraída en la base de datos. En este apartado trataremos de explicar la estructura y el diseño de la aplicación realizada en Android, como de sus interfaces.

2.4.3.1 Patrón Modelo-Vista-Controlador

El patrón Modelo-Vista-Controlador, conocido por las siglas MVC, se caracteriza por ser un patrón arquitectónico que separa el modelo de negocio y la interfaz. El modelo de negocio son los datos y lógica de negocio que implementan la funcionalidad básica de la aplicación. La interfaz es la forma de atender la interacción con el usuario.

Este patrón fue elegido por ser el más extendido en la actualidad y por su excepcional desempeño en todo tipo de proyectos con interfaces interactivas. Aunque esto no es totalmente cierto puesto que muchos consideran que el patrón que mejor se adapta al desarrollo de aplicaciones para Android es el Modelo-Vista-Controlador, ya que la interfaz que haría de Controlador afecta en algunos momentos a la vista. Aun así, exceptuando estos casos, el diseño se adaptará a un MVC por tanto vamos asumir que este es el patrón escogido para esta aplicación.

Para esta aplicación el modelo representa las entidades del dominio, es decir, encapsula los datos que maneja el sistema como la base de datos. En este caso será unas referencias en forma de librería cliente. La vista es la interfaz grafica de usuario donde contendrá elementos con los que el usuario podrá interactuar como botones o menús. Los controladores o la lógica de negocio definirán las operaciones del sistema, realizando las acciones necesarias sobre el modelo.

2.4.3.2 Actividades

Entre el código que nos encontramos, en Android, se usa el termino de Actividad ("*Activity*" en inglés) para denominar a un tipo de clases java que heredan de Activity. Una actividad, como su propio nombre indica, es algo que el usuario puede hacer y en el dispositivo móvil son las pantallas que visualizamos de nuestra aplicación. En Android una actividad es lo mismo que un conjunto de acciones para la iteración directa con el usuario.

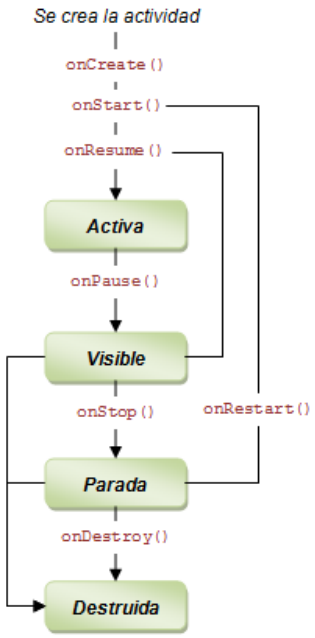


Figura 2.8: Ciclo de vida de una actividad

Una actividad se caracteriza por tener un ciclo de vida. Una aplicación Android corre dentro de su propio proceso Linux. Este proceso es creado con la aplicación y continuará vivo hasta que ya no sea requerido y el sistema reclame su memoria para asignársela a otra aplicación. Como vemos en la imagen anterior (Figura 2.8) la actividad pasa por cuatro estados: Activa, Visible, Parada y Destruida.

- **Activa** (*Running*): La actividad está encima de la pila, lo que quiere decir que es visible y tiene el foco.
- **Visible** (*Paused*): La actividad es visible pero no tiene el foco. Se alcanza este estado cuando pasa a activa otra actividad con alguna parte transparente o que no ocupa toda la pantalla. Cuando una actividad está tapada por completo, pasa a estar parada.
- **Parada** (*Stopped*): Cuando la actividad no es visible. El programador debe guardar el estado de la interfaz de usuario, preferencias, etc.
- **Destruida** (*Destroyed*): Cuando la actividad termina al invocarse el método *finish()*, o es matada por el sistema.

Cada vez que una actividad cambia de estado se van a generar eventos que podrán ser capturados por ciertos métodos de la actividad. En esta aplicación se usará algunos de estos métodos para realizar acciones de comprobación, de almacenamiento o de configuración.

- **onCreate(Bundle)**: Se llama en la creación de la actividad. Se utiliza para realizar todo tipo de inicializaciones, como la creación de la interfaz de usuario o la inicialización de estructuras de datos. Puede recibir información de estado de la actividad (en una instancia de la clase Bundle), por si se reanuda desde una actividad que ha sido destruida y vuelta a crear.

- **onStart()**: Nos indica que la actividad está a punto de ser mostrada al usuario.

- **onResume()**: Se llama cuando la actividad va a comenzar a interactuar con el usuario. Es un buen lugar para lanzar las animaciones y la música.

- **onPause()**: Indica que la actividad está a punto de ser lanzada a segundo plano, normalmente porque otra actividad es lanzada. Es el lugar adecuado para detener animaciones, música o almacenar los datos que estaban en edición.

- **onStop()**: La actividad ya no va a ser visible para el usuario. Ojo si hay muy poca memoria, es posible que la actividad se destruya sin llamar a este método.

- **onRestart()**: Indica que la actividad va a volver a ser representada después de haber pasado por *onStop()*.

- **onDestroy()**: Se llama antes de que la actividad sea totalmente destruida. Por ejemplo, cuando el usuario pulsa el botón de volver o cuando se llama al método *finish()*. Ojo si hay muy poca memoria, es posible que la actividad se destruya sin llamar a este método.

A continuación se muestra el diagrama de clase que representa las actividades de la aplicación. En este diagrama mostramos la estructura que vamos a seguir para su desarrollo.

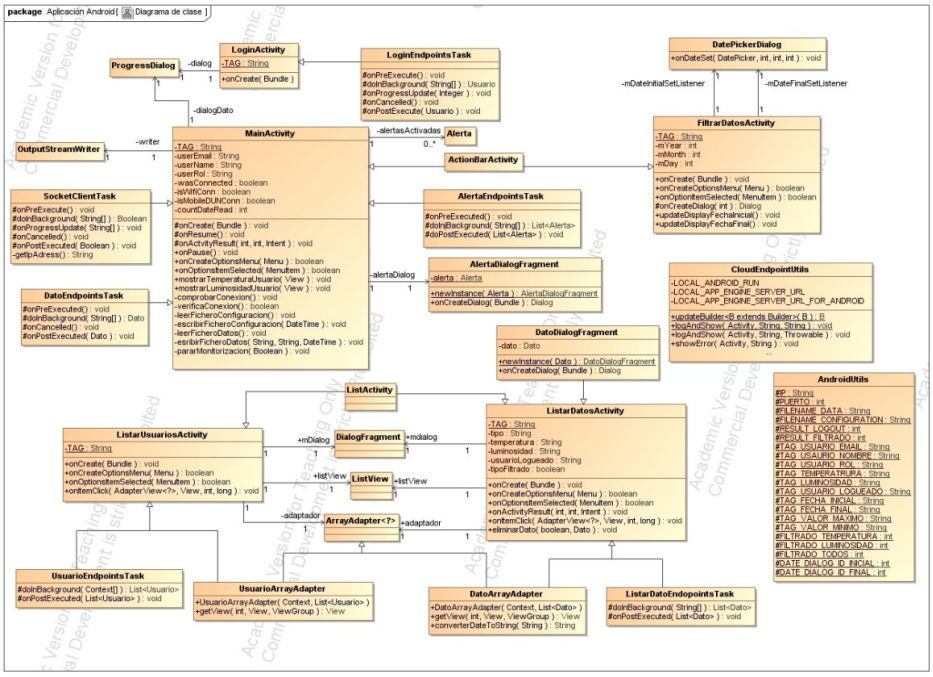


Figura 2.9: Diagrama de clase (Android)

Como vemos en el diagrama, la clase MainActivity será la clase con mayor peso en esta aplicación donde una vez accedido correctamente el usuario guardará la información de esté (email, nombre y rol) para facilitarla en otras acciones. Esta clase alberga varias funciones importantes del sistema, como es la monitorización que es el principal objetivo de este trabajo. Para ello, utilizamos el protocolo TCP/IP basado en socket, esta clase permite crear un cliente que se conectará automáticamente con el servidor. Esto se consigue con la ayuda de una subclase llamada ClientSocketTask, que hereda de la clase parametrizada AsyncTask<*Params, Progress, Result*>, cuya representación en el diagrama es mediante una generalización del *MainActivity*

Estas clases AsyncTask<*Params, Progress, Result*> se encarga de crear una tarea en segundo plano que facilita el uso o la manipulación del hilo de interfaz de usuario (UI thread). Utilizamos esta clase como subclase en diferentes actividades para crear procesos independientes. Para eso tiene tres tipos genéricos, los cuales son "*Params*" que es el tipo de dato que entra en la tarea, "*Progress*" que será el tipo de dato para reflejar el progreso de la tarea, y "*Result*" que es el tipo de dato que devuelve. Asimismo tiene una serie de métodos que se ejecutan en este orden *onPreExecuted(), doInBackground() y onPostExcecuted()*. También otros como *onProgressUpdate()*, para publicar el progreso y *onCancelled()* para procesar cuando el proceso ha sido cancelado.

En el caso del MainActivity, esta subclase crea una tarea que establece la conexión con el servidor Arduino mediante la dirección IP y puerto del servidor. Igualmente, esta tarea almacena temporalmente en un fichero en la memoria interna del dispositivo los cambios de los valores ocurrido durante la monitorización. También gestiona el sistema de alerta por medio de la lista de alerta llamada *listaActivadas*, previamente almacenada en el método *onResume().*

Otras subclase importantes del tipo AsyncTask, son las que utilizamos para realizar peticiones a los servicios web REST alojado en la nube, como LoginEndpointsTask<*String, Integer, Usuario*> en LoginActivity. Esta tarea se encarga de llamar al servicio web para verificar que el email y contraseña que pasamos existen en el sistema.

En este diagrama destacamos dos actividades que realizarán la funcionalidad de mostrar el listado de usuario o historial de datos monitorizados, como son ListarUsuarioActivity y ListarDatosActivity. En las cuales utilizamos clases del tipo ArrayAdapter para establecer listados personalizados de clases. Para el caso del historial de datos también tenemos una actividad para filtrar los datos en el listado

Por último podemos observar dos clases CloudEnpointUtils y AndroidUtils. Esta última contiene constantes utilizadas en la aplicación como por ejemplo, las etiquetas para pasar información de una actividad a otra. En la otra clase se almacena información respecto al servidor web de Google o métodos para construir las peticiones mediante las direcciones URL correspondiente.

2.4.3.3 Interfaces de usuarios

En esta aplicación la interfaz tiene un papel importante para llevar a cabo el proceso principal. Por eso se ha diseñado una interfaz intuitiva para los usuarios y fácil de usar. Como hemos visto en el anterior apartado Android usa actividades. Cada actividad tiene su interfaz *(.xml)* para interactuar con el usuario. El contenido de la interfaz puede varíar según como se gestione dentro de la actividad, tanto informar de los resultados de la acción como los elementos a mostrar.

En este diagrama de navegación reflejamos la acción necesaria para pasar de una actividad a otra, mediante sus interfaces de usuarios.

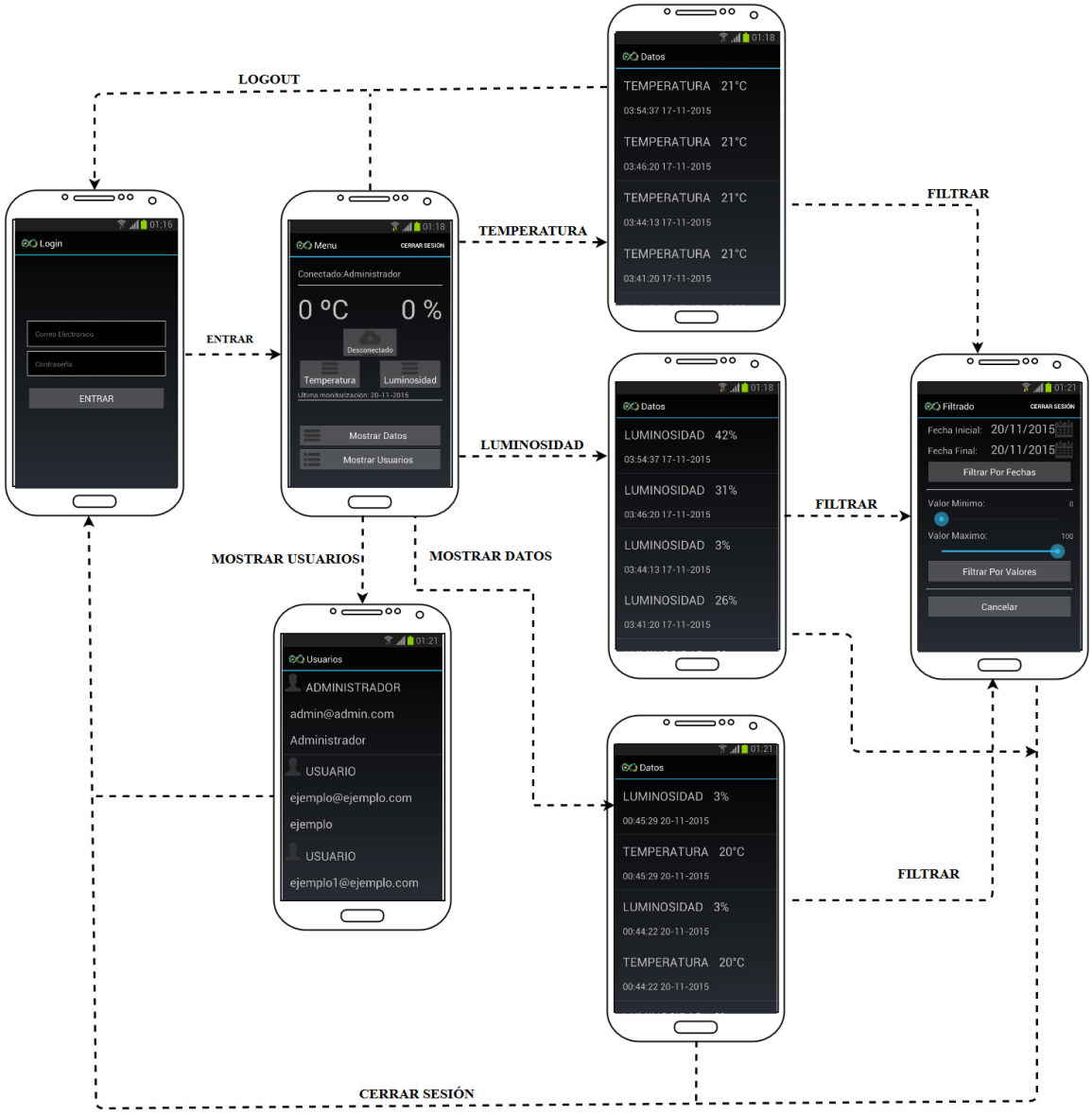


Figura 2.10: Diagrama de navegación de la aplicación Android

Como podemos apreciar en este diagrama se distinguen cuatro interfaces de usuario diferente. A continuación detallaremos los elementos utilizados y su funcionalidad cada una de las vistas.

- **Login**: Esta interfaz es la primera de la aplicación, que nos muestra dos EditText para introducir el email y la contraseña, y un botón para entrar a la aplicación. En esta vista se informará de si la acción ha tenido éxito o no.

- **Menu Principal**: En esta interfaz se encarga de visualizar con dos TextView para monitorizar los valores de temperatura y luminosidad recogidos por los sensores.

  Además, muestra información adicional como el usuario conectado y la fecha de la última monitorización. También tenemos una serie de botones como para activar/desactivar la monitorización (ToggleButton), para acceder a los listados de usuario o datos. En la Figura 2.11 se muestra el aspecto del menú para un usuario registrado como administrador. La diferencia con la vista de un usuario normal, es la visualización de los botones de "*Mostrar Datos*" y "*Mostrar Usuarios*", ya que es un requisito funcional de la aplicación.

Figura 2.11: Interfaz del Menú Principal (Administrador)

- **Listados:** En estos tipos de vistas muestran un listado de usuarios o datos que está relacionado con un ArrayAdapter personalizado estos tipos de entidades. Dependiendo del botón pulsado saldrá un listado u otro, en el caso de los usuarios mostrará el rol, el email y el nombre. Para el caso de los datos, todos o por separado, se verá el tipo de dato seguido de su valor y la fecha en la que fue monitorizada.

- **Filtrado:** A esta vista se accede a partir de un listado de datos. Se encarga de establecer fechas o valores para realizar un filtrado en el listado anterior. Para seleccionar las fechas se mostrará un DatePickerDialog donde podremos cambiar la fecha por defecto (fecha actual). Para elegir el rango de valores utilizamos dos SeekBar para el mínimo y el máximo, cuyos valores predeterminados son 0 y 100 respectivamente.

En todas las vistas excepto el login tiene en común la acción de "*Cerrar Sesión*" en la ActionBar, aunque en las vistas que enseñan un listado no es visible directamente sino haría falta pulsar el botón de menú del dispositivo móvil.

2.4.4 Aplicación Web (Cloud)

La aplicación Web será la parte del sistema que estará alojada en la Cloud, en este caso en el App Engine de Google. Esta aplicación tiene la función de servidor, es decir, manipular las entidades del modelo realizado en el sistema. También contendrá los servicios web (Endpoints) para realizar la comunicación con el cliente móvil Android. Para ello utilizaremos las herramientas y servicios gratuitos ofrecidos por Google.

2.4.4.1 Patrón MVC y capa DAO

Como hemos mencionado en la arquitectura del sistema, para realizar esta aplicación se ha seguido el patrón arquitectónico Modelo-Vista-Controlador. Al igual que la aplicación Android este patrón no permite separar el modelo de negocio y las vistas. El modelo que hemos creado en este trabajo para este sistema será la lógica del negocio de la aplicación.

El componente Controlador de la aplicación Web será una serie de Servlet que procesarán las peticiones del cliente web sobre el modelo, estos serán implementados en Java. Para esta ocasión se han utilizado la tecnología web JSP junto con HTML5 para diseñar las vistas que representarán a la aplicación.

En esta aplicación tendremos una capa de persistencia de datos para acceder a la información de la base de datos, por ello introducimos la capa DAO (Data Access Object). Esta capa nos proporcionará una interfaz común entre la aplicación y el repositorio para el almacenamiento de datos (DataStore). Por eso la arquitectura de esta aplicación Web quedaría de esta forma.

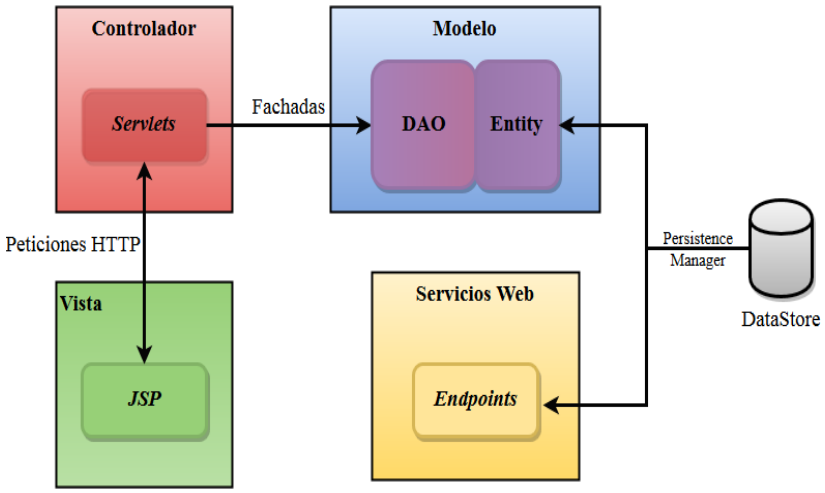


Figura 2.12: Arquitectura de la aplicación Web

El modelo de negocio de la esta aplicación está basada en la programación orientada a objetos. Por tanto, para realizar el acoplamiento entre el usuario y esta capa hemos utilizado el patrón Fachada (Facade). Este patrón nos facilita actuar sobre la persistencia de los objetos Java creado para el modelo, usando JDO (Java Data Object). Para ello incluimos un paquete que implementará esta capa y el patrón Fachada. En este paquete tenemos una clase abstracta que conoce las clases del subsistema, que en este caso son las diferentes entidades de la base de datos. Más adelante profundizaremos más en la capa DAO de la aplicación.

2.4.4.2 Controladores Servlet

Los servlets son módulos que extienden los servidores orientados a petición-respuesta, como los servidores web compatibles con Java. En nuestro caso, el servidor de Google nos proporciona soporte poder llevar a cabo el despliegue de la aplicación. Esto servlets se caracteriza por generar páginas web de forma dinámica a partir de los parámetros de la petición que envié el navegador web.

En esta aplicación abarcamos distintas funcionalidades donde las recogemos mediantes estos servlets. La mayoría de estos servlet procesan las peticiones GET y POST de un cliente y responde con una respuesta, tanto controladores como cliente web. Por ese motivo los separamos en un paquete distinto en el desarrollo del proyecto. Seguidamente explicamos cuales es la función de cada uno.

- **LoginServlet:** Este se encarga de realizar la acción de verificación de los datos, email y contraseña introducida por el usuario, en el sistema para acceder a él. También procesa la acción de cerrar sesión.
- **MenuServlet:** Este servlet procesa información del sistema y del usuario logueado para enviar la petición de respuesta hacia el cliente web.
- **ListarUsuariosServlet:** Obtiene todas las entidades de la clase Usuario para ser enviada y ser mostrada.
- **InsertarUsuarioServlet:** En este servlet se realiza la acción de registrar e insertar un usuario en el sistema. Primero donde se ejecuta la petición mediante la petición GET, y luego en la petición POST se procesa los datos introducidos por el usuario para crear una entidad nueva en la base de datos.
- **EditarUsuarioServlet:** Según el tipo de usuario que realiza la petición, este servlet actualizará la información del usuario logueado como su email, nombre y su contraseña, en el caso de que fuera un usuario normal. En el caso de ser un administrador comprobará si se ha modificado el rol del usuario. También realiza una petición GET para traer al usuario de la base de datos para enviarla posteriormente al cliente web.
- **EliminarUsuarioServlet:** Elimina el usuario seleccionado en el listado por el usuario logueado. Según el tipo de usuario que la realiza se eliminará los

datos y alertas asignados al usuario, en caso de los datos si es el único propietario de ellas se eliminará del sistema.

- **ListarDatosServlet:** Este servlet trata la petición GET para traer todos los datos del sistema, para luego clasificarlos por tipo y enviarlo en la respuesta.
- **FiltrarDatosServlet:** En este servlet se procesará la información recogida por el formulario del listado de datos, para realizar un filtrado de los datos y volver enviarlas. También validará la información introducida por el usuario.
- **EliminarDatosServlet:** Según el tipo de servlet, este servlet, eliminará el dato o su asignación del usuario.
- **ListarAlertasServlet:** Se encarga de recoger todas las alertas asignadas al usuario logueado. También actualiza el estado de la alerta seleccionada si es activada o desactivada por el usuario.
- **InsertarAlertaServlet:** Este servlet procesa la información introducida para insertar una alerta y asignársela al usuario. Dicha información será el tipo de datos y el rango de valores donde se ejecuta dicha alerta. Estos valores también se validarán.
- **EditarAlertaServlet:** Se encargará de actualizar la información de la alerta seleccionada por el usuario. Asimismo realiza una petición GET para traer dicha alerta y enviarla para ser mostrada.
- **EliminarAlertaServlet:** Eliminará la alerta seleccionada por el usuario y desaparecerá su asignación con el usuario.

En todos estos servlet tienen en común que envían un mensaje de vuelta al cliente web, para informar al usuario, dicha información es el estado de la acción tratada por el sistema.

2.4.4.3 Servicios Web

Uno de los objetivos principales del trabajo es la comunicación entre plataformas. En la aplicación web a parte de proporcionar un cliente web, también ofrece servicios web para clientes móviles. En nuestro sistema, el cliente móvil será Android que utilizará estos servicios en el Backend del App Engine para el tratamiento de la información almacenada en el DataStore de Google.

Los servicios web (Endpoints) generado en esta aplicación serán tres que se usarán para acceder a las entidades de la base de datos, los cuales son UsuarioEndpoint, DatoEndpoint y AlertaEndpoint. Dicho endpoints se guardarán en un paquete del proyecto web.

Estos Endpoints son clases que podrán ser generados automáticamente o manualmente a partir de las entidades del modelo de la aplicación. En nuestro proyecto lo hemos realizado automáticamente, esto nos facilita generar una serie de servicios comunes

a cada entidad como son los de insertar una entidad, actualizarla, listar las entidades o eliminar una entidad, entre otros.

A parte de estos servicios en nuestra aplicación ofrecerá otros para satisfacer los requisitos del sistema.

- Verificar datos de un usuario.
- Asignar al usuario un dato.
- Obtener los datos de un usuario según su tipo.
- Asignar un dato a un usuario.
- Filtrar los datos según un rango de valores.
- Filtrar los datos según un rango de fecha.

2.4.4.4 Interfaz Web

La interfaz Web utilizada para esta aplicación tiene un estilo sencillo e intuitivo de usar. Esta es importante para representar los controladores. Para ello hemos utilizado JSP (Java Server Page) que es una tecnología web que nos permite mezclar HTML estático con HTML generado dinámicamente, es decir código Java embebido. Usamos estos JSP ya que se basan en la tecnología Servlet, la cual utilizamos para esta aplicación.

En estas interfaces usamos la estructura de HTML5 junto con CSS para dar estilo a la aplicación. Además, utilizamos en algunas páginas código JavaScript y JQuery para efectuar cambios en las vistas o facilitar al usuario el uso de las tablas.

A continuación enseñamos la navegación principal de la aplicación Web, que seguidamente explicaremos.

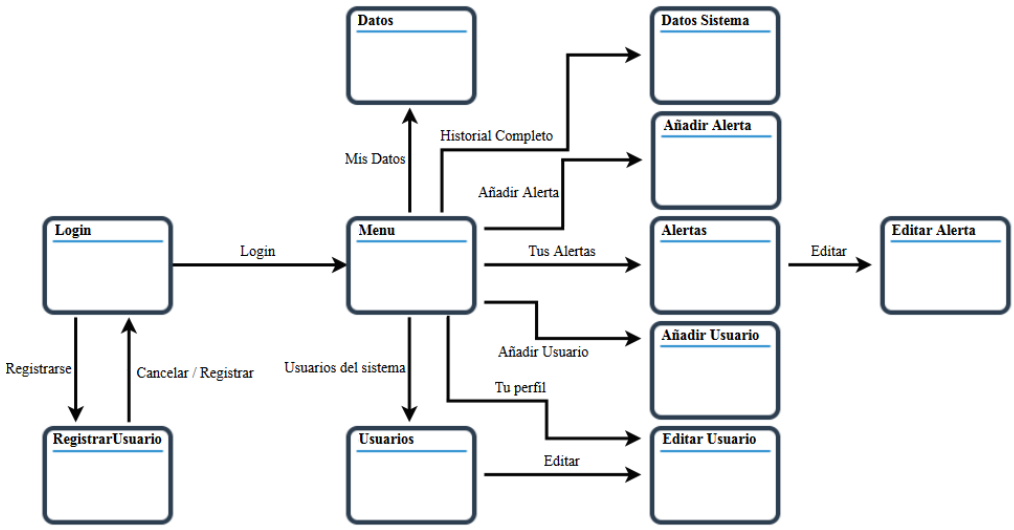


Figura 2.13: Diagrama de navegación Web

En este diagrama podemos observar las diferentes ventanas que se puede encontrar en la aplicación y las acciones llevamos a cabo para ir de una ventana a otra. Existen ventanas que son la misma pero la acción para llegar a ellas son diferente. Una acción que no reflejamos en el diagrama es la de cerrar sesión (Logout) ya que en todas van a la página de Login, menos la propia página y la de registro.

Podemos clasificarla en cinco grupos:

- **Login:** Esta es la primera página al acceder a la aplicación, que te permite introducir tus datos de registro para acceder. También te da la posibilidad de ir a la página para que un usuario se registre en ella.
- **Menu:** En esta página encontramos un contenido con estadísticas del sistema, como un grafico de barras reflejando la cantidad de datos monitorizados en los últimos 30 días. Y diferentes cantidades de datos que representa el total de datos guardados en el sistema, el total de tus propios datos y de cada tipo que tienes monitorizados.
- **Listado:** Es las páginas de tipo listado muestran una tabla con los usuarios, datos o alertas. En ellas te permite realizar acciones como editar o eliminar para cada fila mostrada. Aunque para el caso de los datos solo aparecerá la acción de eliminar. Como podemos ver en al Figura 2.14 en la parte superior derecha del gestor de contenido te proporciona diferentes pestaña para ver los listados según el tipo. También apreciamos un pequeño formulario para filtrar los datos según un rango de valores y fechas.
- **Inserción:** Todas estas muestra un contenido con una serie de campos de texto según la entidad que se vaya a guardar. En el caso de usuario mostramos prácticamente los mismo campo, aunque se llegue de diferentes acciones como son la de registrarse y añadir usuario.
- **Edición:** A esta página le sucede lo mismo que alguna página de inserción, pero dependiendo desde se acceda algunos campos estarán bloqueados para su edición. Estos casos son cuando realizas las acciones de tu perfil o el administrador accede a tus datos desde el listado.

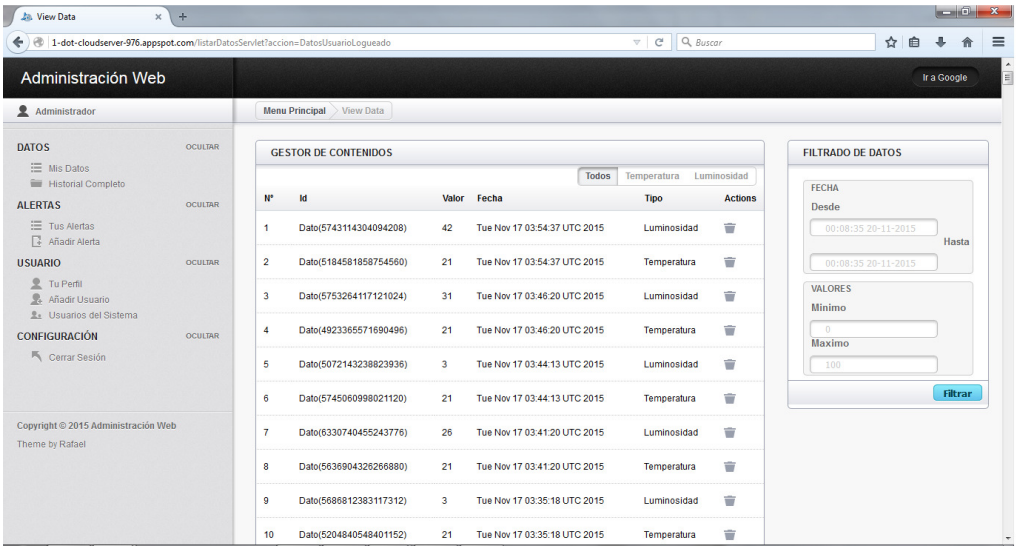


Figura 2.14: Pagina de listado de la aplicación Web

Las estructuras de las páginas están diseñadas de la misma forma, en todas tienen una cabecera donde vemos el nombre de la aplicación Web y justamente abajo una barra secundaria donde vemos el nombre del usuario conectado y una pestaña con el nombre del contenido.

Como vemos en la figura anterior, contiene una barra lateral con las acciones principales de la aplicación, la cual es estática para todas las páginas. Sin embargo, esta barra será el menú de acciones pero dependiendo del usuario que se haya conectado se mostrará más o menos acciones. Por último, en la sección principal veremos las respuestas sobre las acciones de dicha barra de menú.

3. Implementación

En este capítulo hablaremos cómo se ha implementado cada módulo y cómo se ha realizado los procesos para comunicarse entre ellos. Respecto a las aplicaciones Android y Web nos concentramos en los aspectos más relevantes de la implementación.

3.1 Modulo Arduino

Para empezar a implementar este modulo, debemos seguir una serie de pasos para el montaje del circuito mencionado en la parte de diseño. Para ello, realizamos los siguientes:

- Acoplar la WiFi Shield sobre la placa Arduino por los pines de entrada correspondiente y por el puerto SPI.
- Conectar el ánodo (positivo) del led al pin PWM (digital) número 3 del Arduino y al cátodo (negativo) del led a una resistencia de 470 Ω que conectamos con GND.
- Conectar un pin del sensor a un voltaje de 5V y el otro pin del sensor a una resistencia de 10K Ω que a su vez conectamos con GND. Este paso es común para los dos sensores.
- Conectar el pin del sensor de temperatura (NTC) conectado a la resistencia al pin analógico numero 0.
- Conectar el pin del sensor de luminosidad (LDR) conectado a la resistencia al pin analógico numero 1.

Por último, terminamos conectando el cable USB desde el Arduino al PC. En nuestro caso se efectuará de esta forma, aunque también da la posibilidad de conectarse a una alimentación externa.

Una vez finalizado el montaje, procedemos a programar el microcontrolador del Arduino mediante el entorno de programación. En el IDE debemos de estar seguro que se ha conectado al puerto serie al que está conectada nuestra placa, además de seleccionar nuestro modelo Arduino. En Windows, si desconocemos el puerto al que está conectado nuestra placa podemos descubrirlo a través del Administrador de dispositivos (Puertos COM & LPT/ USB Serial Port).

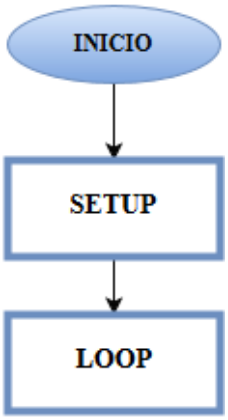


Figura 3.1: Diagrama de flujo del programa principal de Arduino

La estructura básica de programación de Arduino es bastante simple y divide la ejecución en dos partes: *setup* y *loop*. El bloque *setup* constituye la preparación del programa y *loop* es la ejecución. La función *setup()* es la primera en ejecutarse y en nuestro programa realizará funciones de configuración como la WiFi y el servidor. La funcion *loop()* incluye el código a ser ejecutado continuamente, como leer las entradas de la placa, comunicaciones con la red, etc. A continuación iremos desarrollando el código del programa.

3.1.1 Declaración de constantes, variables y del servidor

En nuestro programa declaramos una serie de constantes y variables que utilizaremos a lo largo del código para facilitarnos la programación.

Destacamos 3 bloques de constantes, el primero será para asignar los números de pines a utilizar del Arduino, tanto analógicos como digitales. En nuestro caso serán las constantes TMP1PIN, LDR2PIN y LED1PIN, con valores 0, 1 y 3, respectivamente. Además, tenemos dos bloques de constantes para declarar los parámetros específicos del datasheet del sensor de temperatura, y los limites inferior y superior del valor de lectura del sensor de temperatura.

En la declaración de las variables distinguimos entre las configuraciones de la red a conectar y la declaración del servidor. En dichas configuraciones hemos declarado dos cadenas de caracteres que representan el nombre de la red (SSID) y su contraseña (pass) para nuestro caso en particular, ya que estas pueden variables dependiendo del tipo de red que nos conectemos.

```
WiFiServer server(7000);
```

Para declarar el servidor utilizamos la clase WiFiServer de la librería WiFi, dónde especificamos el puerto por donde escuchará clientes. En nuestro servidor será el puerto 7000. Por otra parte, también declaramos variables enteras o booleanas para almacenar datos temporales o para controlar la salida por el monitor serie.

3.1.2 Setup

Como observamos en la Figura 3.1, la primera función que ejecuta el programa es el *setup()* . En este apartado realizamos configuraciones como asignar pines, configurar la conexión de la red e iniciar el servidor.

En primer lugar, establecemos la velocidad de datos en bits por segundos (baudios) para la transmisión de datos en serie. Luego asignamos al pin 3 como escritura, el cual está conectado el led.

```
Serial.begin(9600);
pinMode(LED1PIN, OUTPUT);

if (WiFi.status() == WL_NO_SHIELD) {
        Serial.println(" WiFi shield no presente ");
        while (true);
}

String fv = WiFi.firmwareVersion();
if ( fv != "1.1.0" ){
    Serial.println("Por favor actualice el firmware ");
}else{
    Serial.print("WiFi.firmwareVersion(): ");
    Serial.print(fv);
}
```

Como vemos en este bloque de código, hacemos dos comprobaciones. La primera sentencia condicional comprueba si esta acoplada la WiFi Shield al Arduino con la constante de estado WL_NO_SHIELD. La segunda sentencia condicional verifica la versión del firmware instalado en la shield. Es importante tener actualizada la versión 1.1.0 ya que nos permitirá realizar las funcionalidades de comunicación de este trabajo.

El siguiente bloque de código muestra como establecemos conexión con la red y dependiendo del valor de la variable status, el programa seguirá su ejecución. Sabremos si se ha conseguido conectar gracias a la constante WL_CONNECTED. Si conseguimos dicha conexión a la red el programa mostrará información de ésta como la dirección IP asignada. Otra forma de saber si la Wifi Shield se ha conectado a la red es mediante el led LINK, el cual se iluminará de verde. Sin embargo si sucede algún problema se iluminará el led ERROR de color rojo.

```
status = WiFi.begin(ssid, pass);
if (status != WL_CONNECTED) {
    Serial.println(" No se pudo obtener conexión wifi ");
    while(true);
```

```
    } else {
        Serial.print(" Conectado a la red ");
        Serial.print(ssid);
        IPAddress ip = WiFi.localIP();
        Serial.print(“ IP Address: ");
        Serial.print(ip);
    }
```

Por último, ejecutamos el método correspondiente para arrancar el servidor Arduino y terminar la función *setup().*

```
     server.begin();
```

3.1.3 Loop

En esta fase del código es donde se ejecuta la función principal del programa del Servidor. Nos basaremos en el diagrama de flujo de la Figura 2.6, en el cuál las dos primeras acciones son de configuraciones que se han realizado previamente en el método *setup().* A continuación comienza un bucle infinito en el que la funcionalidad será extraer un par de datos, uno de temperatura y otro de luminosidad, por cada iteración en la que el cliente esté conectado al servidor.

```
     WiFiClient client = server.available();
```

La primera acción que realiza el servidor es aceptar un cliente disponible. Para ello utilizamos el método *available(),* el cual devuelve un cliente si existiera, en otro caso devolverá 0. Cuando exista un cliente comprobaremos si el cliente está conectado con el método *connected(),* si está conectado se escribirá por pantalla una única vez.

Una vez que el cliente está conectado procedemos a obtener la temperatura y la luminosidad. Para obtener la temperatura primero leemos del pin 0 analógico con el método *analogRead().* Dicho método nos devolverá un valor entre 0-1023, el cuál passamos al método val2temp() junto con las constantes del sensor de temperatura.

```
int valor = analogRead(TMP1PIN);
float tmp = val2tmp(valor, THERMISTOR_T0, THERMISTOR_R0, THERMISTOR_B);
```

Esta función transformará el valor a una temperatura en grados Celsius, para ello se calcula con la siguiente fórmula:

$$R_{TMP} = \frac{1023 \cdot 10000}{valor} - 10000 \quad \tau = \frac{1}{\frac{1}{T_0} + \frac{1}{B}\ln\left(\frac{R_{TMP}}{R_0}\right)} - 273.15$$

Después calculamos la luminosidad leyendo desde el pin 1 analógico el valor del sensor con el mismo método anterior utilizado con la temperatura. El valor devuelto por este método será pasado al método *normalizar()* junto los límites del valor.

```
valor = analogRead(LDR2PIN);
int lum = normalizar(valor, MINLDR, MAXLDR);
```

Esta función normaliza el valor devolviendo el tanto por ciento (%) con respecto a los límites, como vemos en la siguiente fórmula:

$$Lum = \frac{valor}{\text{lim\_ sup} - \text{lim\_}inf} * 100$$

Luego de obtener los valores de temperatura y de luminosidad escribimos en el buffer del servidor para ser enviado al cliente.

```
    server.println(tmp);
    server.println(lum);
```

Para reflejar que se está transmitiendo datos desde la WiFi Shield, empezará a parpadear el led DATA cuando reciba o transmita dato. En nuestro circuito incorporamos un led rojo en el pin 3 digital para plasmar la transmisión de datos con el método *digitalWrite()* cada cierto tiempo de intervalo, en nuestro programa cada segundo escribiremos HIGH para encender y LOW para apagar en el led.

```
    digitalWrite(LED1PIN, HIGH);
    delay(1000);
    digitalWrite(LED1PIN, LOW);
    delay(1000);
```

Y por último, para cerrar la conexión con el cliente utilizaremos el método *stop().* Si el cliente no está conectado se cerrará conexión y se informará en el monitor serie del IDE.

3.2 Modulo Android

En el modulo Android utilizamos muchos elementos visible y funcionales que proporciona esta tecnología. Describiremos los elementos más destacados.

3.2.1 Archivo de configuración Android

El archivo de configuración de Android se llama AndroidManifiest.xml. Situado en la raíz de la aplicación, es un archivo de configuración donde podemos aplicar las configuraciones básicas de nuestra aplicación. Su configuración puede realizarse a través de una interfaz grafica, pero es recomendable conocer la sintaxis ya que en muchas ocasiones será más fácil y rápido hacerlo desde el propio xml.

La primera configuración podemos establecer son las versiones soportadas, con esta etiqueta `<uses-sdk>`. En nuestro proyecto hemos usado como la versión 8 y 21 como mínima y máxima, respectivamente.

```
<uses-sdk android:minSdkVersion="8" android:targetSdkVersion="21"/>
```

Otra configuración, son los permisos que el usuario debe aceptar al instalar. Dado que en nuestra aplicación hacemos uso de Internet y tenemos que acceder al estado de la red y de la Wifi del dispositivo tendremos que usar estos permisos:

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
```

En este archivo nos permite configura aspecto sobre la aplicación en general como el icono, el nombre y la temática. Esto se configura en la etiqueta `<application>` y sus correspondientes atributos como vemos a continuación.

```
<application
       android:icon="@drawable/ic_launcher"
       android:label="@string/app_name"
       android:theme="@style/Theme.AppCompat">
```

Dentro de esta etiqueta declaramos todas las actividades que contiene la aplicación, y cual de ella será la actividad de arranque. Esta actividad se considera una actividad implícita ya que no es llamada desde otra. En nuestra aplicación utilizaremos las actividades de forma explícita. Esto se introduce de la siguiente forma en el archivo xml.

```
<activity android:name=".MainActivity" android:label="@string/app_name" />
<activity android:name=".ListarDatosActivity" android:label="@string/app_name" />
<activity android:name=".ListarUsuariosActivity" android:label="@string/app_name" />
<activity android:name=".InsertarUsuarioActivity" android:label="@string/app_name" />
<activity android:name=".FiltrarDatosActivity" android:label="@string/app_name" />
<activity android:label="@string/app_name" android:name=".LoginActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
</activity>
```

3.2.2 Actividad con resultado

Como hemos visto Android se basa en actividades, la cuales pasan de una a otra según la aplicación. En nuestra ocasión llevamos a cabo el paso de actividades explícitamente, por tanto debemos de introducirla en el código. Esto se consigue a través de los *"Intent"* de Android que nos permite además enviar información de una actividad a otra.

Estos Intents se crean con el contexto de la actividad actual y la actividad a la que se quiere desplazar. Luego, añadimos a este objeto intent la información que se desea transmitir y posteriormente comenzar o activar la otra actividad.

```
Intent intentListarDatos = new Intent(MainActivity.this,
ListarDatosActivity.class);

intentListarDatos.putExtra(AndroidUtils.TAG_TEMPERATURA, "1");
intentListarDatos.putExtra(AndroidUtils.TAG_LUMINOSIDAD, "0");
intentListarDatos.putExtra(AndroidUtils.TAG_USUARIO_LOGUEADO, userEmail);

startActivityForResult(intentListarDatos, 0);
```

En este bloque de código mostramos cuando se pulsa el botón de mostrar las temperaturas. En esta caso pasamos información con el método `putExtra(name,value)`, que pasan dos cadenas, una con el nombre específico del metadado asignado para enviar y la otra con el valor del dato a enviar. En esta situación se envía tres datos, en dos de ellos reflejan si se desea el listado de temperatura o luminosidad, pasando una cadena con el valor 1 o 0, con la petición del usuario. Además, el email del usuario conectado.

Los objetos Intent se pueden empezar de dos formas diferentes, una directa y otra con resultado. En la forma directa simplemente comienza la actividad cuando se ejecuta este método `startActivity(intent)`. De la otra forma la actividad que inicio la segunda actividad esperará un resultado de esta actividad, que nos permitirá gestionar las siguientes acciones dependiendo de este resultado, con el método `startActivityForResult(intents, ResultCode)`.

En el siguiente código mostramos la secuencia que sigue el programa para pasar de ListarDatosActivity a FiltrarDatosActivity cuando pulsamos la opción de "Filtrar datos" del menú de opciones.

ListarActivity.java

```
Intent intentFiltrarDatos = new Intent(ListarDatosActivity.this,
FiltrarDatosActivity.class);

// Iniciamos la actividad segun el tipo de filtrado solicitado
if(temperatura.equals("1")){
      startActivityForResult(intentFiltrarDatos,
      AndroidUtils.FILTRADO_TEMPERATURA);
}else if(luminosidad.equals("1")){
```

```
        startActivityForResult(intentFiltrarDatos,
                               AndroidUtils.FILTRADO_LUMINOSIDAD);
}else{
        startActivityForResult(intentFiltrarDatos, AndroidUtils.FILTRADO_TODOS);
}
```

FiltrarDatoActivity.java

```
Intent intent = getIntent();

// Enviamos los datos que queremos mandar a la actividad principal.
String strFecha = "00:00:00 " + textViewFechaInicial.getText().toString();
intent.putExtra(AndroidUtils.TAG_FECHA_INICIAL, strFecha);
strFecha = "00:00:00 " + textViewFechaFinal.getText().toString();
intent.putExtra(AndroidUtils.TAG_FECHA_FINAL, strFecha);

// Establecemos el resultado, y volvemos a la actividad principal.
// La variable que introducimos en primer lugar "RESULT_FILTRADO" es
// de la clase AndroidUtils.
setResult(AndroidUtils.RESULT_FILTRADO, intent);

// Finalizamos la Activity para volver a la anterior
finish();
```

En la actividad del filtrado llamamos al método `setResult(ResultCode, Intent)` con código resultado y devolvemos el intent con los datos que se desee, en este caso el rango de fechas a filtrar, para que se llame implícitamente el método `onActivityResult(RequestCode, ResultCode, Intent)` en `ListarDatosActivy.java` .

```
if (resultCode == AndroidUtils.RESULT_FILTRADO){
   // Recogemos la información devuelta por la segunda actividad.
   String fechaInicial = data.getStringExtra(AndroidUtils.TAG_FECHA_INICIAL);
   String fechaFinal = data.getStringExtra(AndroidUtils.TAG_FECHA_FINAL);
   String valorMinimo = data.getStringExtra(AndroidUtils.TAG_VALOR_MINIMO);
   String valorMaximo = data.getStringExtra(AndroidUtils.TAG_VALOR_MAXIMO);

   // Si las fechas no son nulas, el tipo de filtrado es por fecha (true)
   if(fechaInicial != null && fechaFinal != null){
      tipoFiltrado = true;
   }
   // Si los valores no son nulos, el tipo de filtrado es por valor
   if(valorMinimo != null && valorMaximo != null){
      tipoFiltrado = false;
   }
   // Y tratamos el resultado en función del tipo de filtrado solicitado
   switch (requestCode) {
      case AndroidUtils.FILTRADO_TEMPERATURA:
            listarDatos(fechaInicial, fechaFinal, valorMinimo, valorMaximo);
            break;
      case AndroidUtils.FILTRADO_LUMINOSIDAD:
            listarDatos(fechaInicial, fechaFinal, valorMinimo, valorMaximo);
            break;
      case AndroidUtils.FILTRADO_TODOS:
            listarDatos(fechaInicial, fechaFinal, valorMinimo, valorMaximo);
           break;
   }
```

3.2.3 Elementos visuales

En la aplicación Android utilizamos muchas clases para la interfaces de usuario, entre otros:

- **ArrayAdapter**: Esta clase es de gran ayuda para generar los ListView de las actividades que muestra listado de elementos con vistas personalizadas en las correspondientes actividades.
- **ToggleButton:** Este tipo de botón se utiliza para activar o desactivar la monitorización. Cuando el indicador se ilumina significará que está el proceso en marcha, en el caso contrario la monitorización estará parada.
- **ProgressDialog:** Este tipo de dialogo lo utilizamos para los procesos de espera de la aplicación mediante el estilo Spinner junto un mensaje informando al usuario.

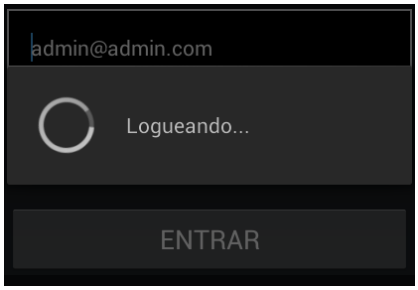


Figura 3.2: ProgressDialog con estilo Spinner

- **DialogFragment:** Esta clase nos permite crear un dialogo que puede mostrar un tiluo, botones y un mensaje, donde el usuario puede interactuar con él. Nuestra aplicación la usamos para eliminar datos o informar de una alerta.

- **DatePickerDialog:** Esta clase nos permite crear un cuadro con un DatePicker donde el usuario puede selecciona una fecha, como las fechas de la actividad de filtrado.

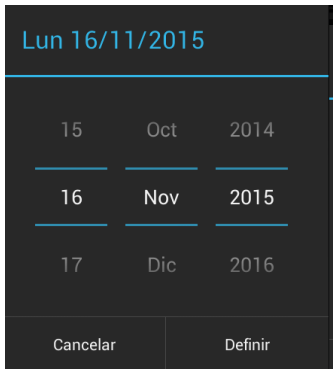


Figura 3.3: DatePickerDialog

- **Toast:** Esta la usamos para informar al usuario mediante mensajes como "Monitorización Cancelada" o "Accediendo" junto el nombre del usuario logueado.

3.3 Modulo Cloud

En este trabajo, la plataforma Cloud de Google es donde alojaremos la aplicación Web y el servidor Web, gracias a Google App Engine. Para empezar a desarrollar el proyecto en al nube necesitamos proporcionar una cuenta de Google para acceder al Google Developers Console, que te permite gestionar tus proyectos en la nube y ofreces muchos servicios para creación o testeo de tu aplicación.

Cuando crea un proyecto en la consola de Google te proporciona un identificador y un número asociado dicho proyecto.

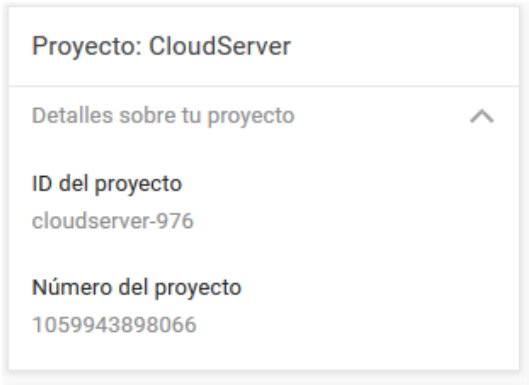


Figura 3.4: ID y Número del proyecto

Para acceder a tu aplicación te ofrece dos direcciones, en nuestro caso:

- https:// 1-dot-cloudserver-976.appspot.com
- http://cloudserver-976.appspot.com

3.3.1 Archivos de configuración App Engine

En el proyecto web se generan varios archivos de configuración, uno de ellos para asociar la aplicación web con la plataforma Cloud de Google, como *appengine-web.xml*. Además en la aplicación web se creará un archivo de configuración para desplegar, que se llamará *web.xml*.

Para poder desplegar la aplicación debemos asignar en el archivo de *appengine-web.xml* el identificador del proyecto creado en la Google Developers Console. Esto se consigue añadiendo dentro del archivo una etiqueta `<application>` con el id del proyecto. También tenemos que asignar con la etiqueta `<version>` el identificador de la última versión del código de la aplicación. Y una etiqueta `<threadsafe>` que define si App Engine puede enviar varias solicitudes al mismo tiempo a un servidor web determinado.

```
<appengine-web-app xmlns="http://appengine.google.com/ns/1.0">
  <application>cloudserver-976</application>
  <version>1</version>
```

```
  <threadsafe>true</threadsafe>

  <sessions-enabled>true</sessions-enabled>
  <async-session-persistence enabled="true" />

</appengine-web-app>
```

App Engine incluye una implementación de sesiones, mediante la interfaz de sesión de servlet. La aplicación almacena datos de sesión en el almacén de datos (DataStore) de App Engine para la persistencia, y también utiliza *memcache* para la velocidad. Al igual que con la mayoría de los otros contenedores de servlets, los atributos de la sesión que se establecen con *session.setAttribute()* durante la solicitud se conservan en el extremo de la solicitud. Por ese motivo incluimos las dos últimas etiquetas del archivo.

El otro archivo de configuración importante es el *web.xml,* donde establecemos los patrones url para acceder a los servlet creado en la aplicación. También nos permite decir cuál será la página de bienvenida de la aplicación web.

3.3.2 Entidades con JDO

Las entidades de la base de datos del sistema son creadas con JDO para almacenar objetos de datos Java simple en el almacén de datos (DataStore). Cada objeto que se hace persistente con el PersistenceManager se convierte en una entidad en el almacén de datos. Se utiliza anotaciones para decirle al JDO como almacenar y crear instancias de sus clases de datos.

Declaramos una clase Java como capaz de ser almacenada y recuperada desde el almacén de datos con la anotación `@PersistenceCapable`. Los campos de la clase de datos que se van a almacenar en el almacén de datos deber ser declarados como campos persistentes con la anotación `@Persistent`.

En nuestra aplicación tenemos tres entidades cuyos objetos Java están declarado como persistente: Usuario, Dato y Alerta. Toda clase persistente necesita tener uno y solo un campo dedicado a la clave primaria para ser identificada en el almacén de datos, para ello utilizamos la anotación `@PrimaryKey.` La clave primaria puede ser de varios tipos, en nuestro caso será una cadena para guardar el email para entidad Usuario y un tipo *Key,* que te rellena automáticamente por JDO con un valor único a través de todas las demás instancias de la clase cuando el objeto se guarda en el almacén de dato, para los identificadores de tipo *long* de las entidades Dato y Alerta. Para que realice este asignado automatico de clave hay que añadir la siguiente anotación:

```
@Persistent(valueStrategy=IdGeneratorStrategy.IDENTITY)
```

3.3.3 Capa de persistencia

En este sistema es necesario acceder a un subsistema, como es el almacén de datos de la Cloud. Por este motivo necesitamos una capa de persistencia (DAO) que nos permita tener acceso a las entidades del DataStore. Esto se consigue con la clase *PMF.java* que crea una instancia del tipo objeto `PersistenceManagerFactory` como vemos en la clase.

```
public final class PMF {

    private static final PersistenceManagerFactory pmfInstance =
       JDOHelper.getPersistenceManagerFactory("transactions-optional");

    private PMF() {}

    public static PersistenceManagerFactory get() {
       return pmfInstance;
    }
}
```

En esta capa implementamos el patrón Fachada (o *Facade*) que nos permite crear un interfaz para reducir la complejidad de la estructura del entorno de programación. Este patrón consta de una clase genérica abstracta llamada `AbstractFacade<T>` y un conjunto de subclases que heredan de esta, en nuestro caso son tres: *UsuarioFacade, DatoFacade y AlertaFacade*. Esta clase abstracta nos proporciona una serie de métodos en los que usará este objeto persistente los métodos facilitado por la interfaz JDO, como `makePersistent(entity)` que creará una entidad o la editará si ya existe, `deletePersistent(entity)` que eliminará una entidad o `getObjectById(entityClass, id)` que recuperará la entidad con la clave primaria de está.

En estos facades también se usan consultas, o *query*, para obtener colecciones de entidades. Estas consultas se crean de la siguiente forma:

```
Query query = getPersistenceManager().newQuery(entityClass);
```

En nuestras subclases utilizaremos estas consultas para filtrar las entidades de la colección y ordenarlo por algún atributo. A continuación mostramos ejemplo de estas consultas.

```
query = getPersistenceManager().newQuery(Dato.class);
query.setFilter("valor <= " + valueMaxString + " && valor >= " +
valueMinString);


query = getPersistenceManager().newQuery(Dato.class);
query.setOrdering("fecha DESC");
```

3.3.4 Aplicación Web

En nuestra aplicación Web como hemos mencionado en el apartado de diseño se realiza mediante Servlet y JSP. Cada uno de los servlet de nuestra aplicación representa una funcionalidad que llevarán a una página web de la aplicación. Estos servlet de esta aplicación Web se asocian a los requisitos del sistema.

En esa aplicación, también utilizamos otras API de Google, como GoogleChart. Esta API nos proporciona métodos para cargar un gráfico en la página web gracias a JavaScript. Utilizamos esta API en el contenido del menú principal (*menu.jsp*) para mostrar un grafico para cada tipo de datos con la media de los valores monitorizados en los últimos 7 días. Esto se consigue con la respuesta de *MenuServlet.java*.

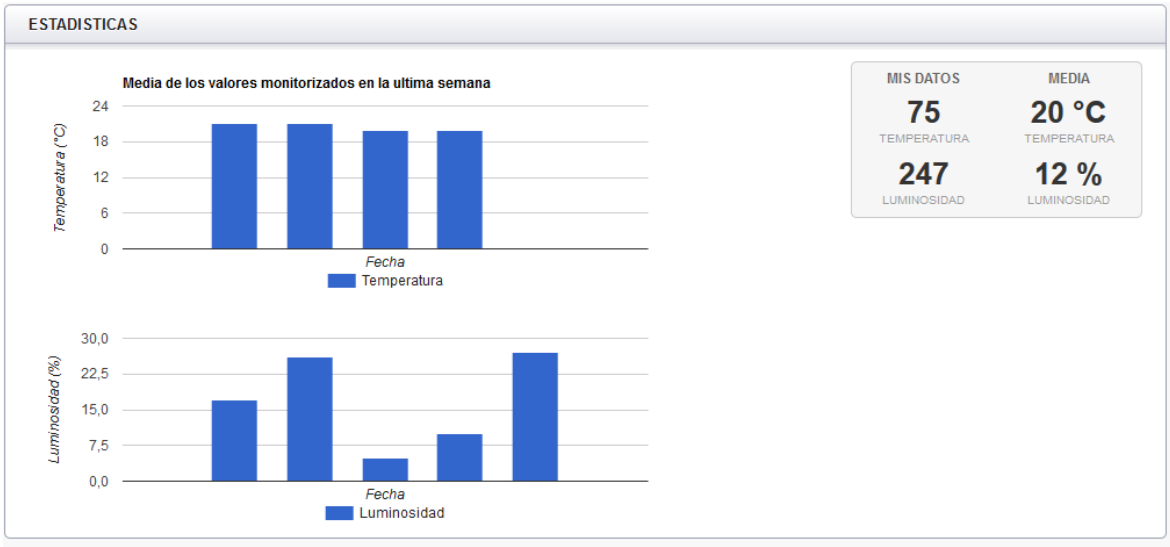


Figura 3.5: Grafico con GoogleChart

También mostramos el número de datos monitorizados durante la última semana para cada tipo de dato, temperatura y luminosidad, junto con la media de estos valores.

3.4 Comunicación entre los módulos Arduino y Android

En este apartado implementaremos uno de los objetivos principales de este sistema, que es el proceso de monitorización. Este proceso se lleva a cabo en realizar comunicaciones entre el dispositivo WiFi acoplado en Arduino y el dispositivo móvil Android. Como hemos comentando en el diseño este utilizará el protocolo TCP/IP basado en socket, donde el Arduino será el servidor TCP que ofrece servicios sobre la obtención de valores de los sensores y el Android es el cliente que obtiene estos servicios.

El proceso tiene varias fases, las del servidor ya fue explicada en la implementación del modulo Arduino, pero en el cliente será donde se encuentre el proceso de monitorización en sí. Este proceso se realiza en la actividad principal (*MainActivity.java*) de la aplicación móvil. Para su construcción utilizaremos métodos como *onResume()*, o tareas AsyncTask de Android como SocketClientTask.

En los siguientes apartados se detallará cada una de las fases por la que pasa este proceso.

3.4.1 Configuración previa

La fase de configuración se encarga de preparar las correspondientes variables que utiliza este proceso. Este proceso puede empezar de varias formas, por lo que estas configuraciones se aplicarán en todos los casos. El caso más importante se encuentra en el método *onResume()*.

La primera acción que ejecutamos es la lectura del archivo guardado en memoria interna, para ello llamamos el método `leerFicheroConfiguracionMemoriaInterna()`, que nos actualizará por pantalla el TextView que contendrán la fecha de la ultima monitorización. También creamos la variable para escribir sobre el fichero de datos en memoria interna.

Lo siguiente que configuramos es la lista de alerta activadas (`alertasActivadas`) que se actualizará con la tarea `AlertaEndpointsTask`, que se le pasará el email del usuario conectado para traer las alertas del DataStore que tiene activada dicho usuario.

La siguiente acción significativa es la verificación de la conexión, que actualizaremos las variables booleanas `isWifiConn` y `isMobileDUNConn`, que nos indicará si la conexión WiFi del dispositivo está conectado. Esta acción la modulamos con el método `verificarConexion()` que mostramos a continuación:

```
private void verificarConexion(){
   connMgr = (ConnectivityManager) getApplicationContext()
                           .getSystemService(Context.CONNECTIVITY_SERVICE);

   networkInfo = connMgr.getNetworkInfo(ConnectivityManager.TYPE_WIFI);
   isWifiConn = networkInfo.isConnected();

   networkInfo = connMgr.getNetworkInfo(ConnectivityManager.TYPE_MOBILE_DUN);
   isMobileDUNConn = networkInfo.isConnected();
}
```

En este trabajo utilizaremos dos tipo de WiFi para conectarse al servidor Arduino, dependiendo de la red que está conectada. Como vemos en el método anterior obtenemos el servicio del sistema para conseguir el objeto `ConnectivityManager` que responde a consultas sobre el estado de la conectividad de red. Este servicio se realiza gracias al permiso *`ACCESS_NETWORK_STATE`* del archivo de configuración (*Manifiest.xml*) de Android.

Por último, tenemos unas sentencias de condicional con tres casos distinto, que dependerá de las variables actualizada anteriormente. El primer caso ocurre cuando existe una conexión WiFi y si entra por primera vez o estaba conectado. Este ultimo estado se refleja con el booleano `wasConnected` que por defecto será **`false`** e irá actualizando durante la ejecución de la aplicación. Cuando ocurre esto significará que la monitorización que se ejecutará automáticamente la tarea `SocketClientTask`.

En el siguiente caso sucederá cuando la conexión WiFi este activada y no estaba en ejecución la monitorización al realizar cualquier acción provocada por el usuario o por el propio móvil. En caso contrario se mostrará por pantalla un mensaje de `Red Wifi Desconectada`. En todos los casos se llamará al método `actualizarEstado(Boolean, Integer)` que pasará por el primer parámetro un booleano con el estado de la conexión y un entero con la constantes `View.`***`VISIBLE`*** o `View.`*`INVISIBLE`*, para reflejar un mensaje por pantalla.

3.4.2 Ejecución

Una vez realizada la configuración previa, se ejecutará la tarea `SocketClientTask` que se efectuará la conexión con el servidor Arduino y recibirá la información transmitida por este. Es decir, la temperatura y luminosidad de los sensores.

En primer lugar, entrará en el método `onPreExecute()` que ocultará cualquier mensaje mostrado anteriormente. Seguidamente, se ejecutará el método principal de la tarea `doInBackground()` donde se conectará con el servidor creando un objeto `Socket` para llevar a cabo la comunicación. En el caso de que no estuviera el servidor conectado saltará esta excepción `ConnectException` y devolverá **`false`**, en el escenario de éxito devolverá **`true.`** Si conseguimos conectarnos la tarea entrará en un bucle infinito, donde leeremos el buffer de datos del socket y se monitorizarán con el método `publishProgress()`, y no terminará a no ser que se cancele dicha tarea, manualmente con el `ToggleButton` o por cualquier otra acción de cancelación.

En el caso de que se cancele la tarea llamará al método `onCancelled()` que enseñará un notificación del tipo `Toast` con el mensaje `Monitorización Cancelada` y terminará la tarea. Para el caso de que se paré manualmente, el bucle finaliza y ejecutará el método `onPostExecute(Boolean)`. Este método recoge la variable devuelta por el método principal y dependiendo de su valor informará al usuario de la finalización de la tarea. Si el resultado es **`false`** significará que el servidor no está conectado y por tanto notificará con un mensaje de `Servidor No Conectado`, en caso positivo, se notificará con un mensaje `Fin Lectura`.

3.4.3 Procedimiento de almacenamiento

Durante la ejecución de este proceso se realiza un procedimiento para guardar los datos monitorizados. Para ello, la tarea guardará en un fichero de la memoria interna del dispositivo Android los datos cambios de temperatura y luminosidad durante la monitorización. Esto se consigue guardando el estado anterior de las variables y su posterior comparación con el siguiente valor monitorizado.

Cuando sucede este cambio, actualizará la variable y llamará al siguiente método `escribirDatosFicheroMemoriaInterna(String, String, DateTime)` para escribir en el fichero. Se le pasarán la siguiente información por parámetros el tipo de datos, el dato leído y la fecha de monitorización. En este método procesará y escribirla en el fichero `"DatosLeidos.txt"` con la siguiente estructura:

```
Temperatura 22 2015-11-15T17:47:04.305+01:00
Luminosidad 50 2015-11-15T17:47:04.339+01:00
                ...
```

3.4.4 Sistema de alerta

Como hemos visto en el modelo de este sistema, se encuentra la entidad Alerta asociada a un usuario. El usuario puede crear y activar varias alertas desde la aplicación Web. El objetivo de estas alertas es parar la monitorización cuando rebase los valores de la alerta activada.

Figura 3.6: DialogFragmet de la alerta

Este sistema se lleva a cabo en el proceso principal de la tarea. Cada vez que lee y monitoriza los valores, inmediatamente comprueba en la lista de alertas activadas si el valor esta fuera del rango de la alerta del tipo de dato correspondiente. Cuando sucede esta caso aparecerá un cuadro de dialogo (`DialogFragment`) con la información de la alerta lanzada.

Como vemos en la Figura 3.5**,** el sistema ofrece dos opciones, *"Parar"* o *"Seguir"*. Cuando se pulsa cualquiera de las opciones, el dialogo recogerá dicha opción y llamará al método `pararMonitorizacion(Boolean)` que dependiendo de la opción pulsada realizará el proceso de terminación y posterior almacenamiento de datos, o se reiniciará la monitorización.

El proceso de terminación se encarga de cerrar el escritor (`writer`) del fichero de datos, leyendo el fichero para empezar el proceso de almacenamiento con el método `leerDatosFicheroMemoriaInterna()` y actualizar el fichero de configuración con la fecha actual `escribirFicheroConfiguracionMemoriaInterna(DateTime)`.

3.5 Comunicación entre los módulos Android y Cloud

La comunicación entre el dispositivo móvil y la plataforma Cloud se consigue mediante servicios Web REST. Asimismo, dicha comunicación existirá un cliente y un servidor. En este sistema el móvil Android tendrá el rol de cliente y la Cloud de Google alojará el servidor. Estas comunicaciones tienen como objetivo almacenar y recuperar datos del almacén de datos (DataStore) .

3.5.1 API Endpoints

Los endpoints es la solución de Google para ayudar a crear una API pública para la aplicación de App Engine. Cloud Endpoints también dispone de herramientas para generar librerías de cliente para Android, las cuales en nuestro proyecto nos facilita la tarea de integrar la funcionalidad de la aplicación backend en la aplicación móvil en Android.

En nuestro servidor tendremos un Endpoint por cada entidad del modelo, por se llamarán UsuarioEndpoint, DatoEndpoint y AlertaEndpoint. Estos utilizan anotaciones para definir propiedades y comportamiento en las configuraciones, los métodos y los parámetros de los endpoints. Además, los endpoints generados tienes los métodos para realizar un CRUD con las correspondientes entidades. Estos métodos son los equivalente a los métodos HTTP: GET (para la devolución de información), POST (para insertar), PUT (para actualizar) y DELETE (para eliminar).

Estas API de Google Cloud Endpoints se configuran en el archivo de configuración *web.xml* como vemos a continuación.

```
<servlet>
  <servlet-name>SystemServiceServlet</servlet-name>
  <servlet-class>com.google.api.server.spi.SystemServiceServlet</servlet-class>
  <init-param>
   <param-name>services</param-name>
   <param-value>app.cloud.endpoints.AlertaEndpoint,
                  app.cloud.endpoints.DatoEndpoint,
                  app.cloud.endpoints.UsuarioEndpoint</param-value>
  </init-param>
</servlet>
<servlet-mapping>
   <servlet-name>SystemServiceServlet</servlet-name>
   <url-pattern>/_ah/spi/*</url-pattern>
</servlet-mapping>
```

Las aplicaciones envían sus solicitudes a *"/_ah/api"* y el backend Endpoints maneja estas peticiones en la ruta *"/_ah/spi"*. El *"_ah*" es el namespace reservado por AppEngine. Esto nos permite componer la URL para acceder a nuestro API. El esquema, en nuestro caso, es el siguiente:

*https://cloudserver-976.appspot.com/_ah/api/**ApiName/Version/Path**?aram1&...&paramN*

Donde:

- **ApiName**: es el nombre del API que corresponde con el atributo `@Api(name="ApiName")`.
- **Version:** es la versión de nuestra API.
- **PathMethod:** es el path del método solicitado que corresponde con el atributo `@ApiMethod(path ="PathMethod")`.
- **Param1&..&ParamN:** son los parámetros que le pasamos al método.

En nuestros Enpoints construimos varios métodos para satisfacer los servicios del cliente Android. A continuación mostramos un ejemplo de cómo sería un método de los Endpoints. Este ejemplo es método *loginUsuario()* que está en el *UsuarioEnpoint.java*.

```
@ApiMethod(name="loginUsuario", path="loginUsuario",
           httpMethod = HttpMethod.GET)
public Usuario loginUsuario(@Named("email") String email, @Named("password")
String password) {

       PersistenceManager mgr = getPersistenceManager();
       Usuario usuario = null;
       try {
             usuario = mgr.getObjectById(Usuario.class, email);
             if(!usuario.getPassword().equals(password)){
                   usuario = null;
             }

       } catch (javax.jdo.JDOObjectNotFoundException ex) {
             usuario = null;
       } finally {
             mgr.close();
       }

       return usuario;
}
```

Como vemos anotamos `@ApiMethod` al método para configurar algunos atributos como `name="loginUsuario"` que será utilizado para utilizarlo en las librerias del backend. También hay otro atributo para el tipo de petición que responde denominado `httpMethod = HttpMethod.GET`.

Dentro del cuerpo del método debemos utilizar los métodos de la interfaz JDO que nos proporciona mediante el objeto `PersistenceManager`, el cual se obtiene de la clase *PMF.java.*

3.5.2 Tareas AsyncTask

En la aplicación Android utilizamos las tareas asíncronas para crear procesos en segundo plano. En nuestro caso usaremos las clases *AsyncTask<Param, Progress, Result>* para acceder a los servicios Web REST ofrecido por nuestro Endpoints. Es decir, para insertar, obtener colecciones o eliminar entidades del almacén de datos.

La forma de crear estos endpoints en estas tareas son a través de las librerías de cliente generadas de la aplicación Web. Para ello debemos obtener el objeto Endpoint como vemos en el sigue código.

```
Datoendpoint.Builder endpointBuilder = new Datoendpoint.Builder(
                                       AndroidHttp.newCompatibleTransport(),
                                       new JacksonFactory(),
                                       new HttpRequestInitializer() {
                                             public void initialize(HttpRequest
                                             httpRequest) { }
                                       });

Datoendpoint endpoint =
CloudEndpointUtils.updateBuilder(endpointBuilder).build();
```

Este código refleja la obtención del Datoendpoint, el cual podremos usar para llamar a los servicios de dicho endpoint. Por tanto cada vez que queramos usar los endpoints debemos de crearlo de esta forma.

3.5.3 Proceso de almacenamiento

Este proceso se ejecuta, en *MainActivity,java,* justamente después del proceso de monitorización para almacenar los datos guardados en el fichero de memoria interna. Este proceso empieza llamando al método `leerDatosFicheroMemoriaInterna()` donde leeremos la información de los datos escrito del fichero `"DatosLeidos.txt"`. Dicho método creará una tarea asíncrona para insertar cada dato leído en el almacén de datos hasta que no haya ninguno.

Como hemos comentando en el apartado anterior, utilizaremos las clases AsyncTask para realizar almacenar el objeto en el DataStore. Para este proceso creamos la clase `DatoEndpointsTask extends AsyncTask<String, String, Dato>` en la que pasamos cuatro parámetros de tipo *String* y devuelve un objeto de tipo Dato. Estos parámetros son recogido por el método principal de la tarea con una estructura tal que asi, `doInBackground(String... params)` dicha estructura representa un array de tipo *String* de tamaño como parámetro hayan sido introducido al crear la tarea. Por lo que el orden de cada elemento del array quedaría de la siguiente forma:

- **params[0]:** Tipo de dato a insertar.
- **params[1]:** Valor monitorizado del tipo de dato.
- **params[2]:** Fecha de la monitorización.
- **Params[3]:** Email del usuario logueado.

El método `doInBackground()` empieza obteniendo dos endpoints: Datoendpoint y UsuarioEndpoint. DatoEndpoint para insertar el dato y UsuarioEndpoint para asignar el dato al usuario logueado. Una vez creados, tratamos los parámetros y creamos un objeto Dato con dichos parámetros. Despues de esto ejecutamos los servicios de los correspondientes endpoints.

```
// Insertamos el dato en el DataStore
Dato newDato = endpointDato.insertDato(dato).execute();

// Asignamos el dato nuevo al usuario logueado
Usuario usuarioConectado = endpointUsuario.asignarDato(params[3],
                                        newDato.getId().getId()).execute();
```

En esta ocasión los método `onPreExecute()` y `onPostExecute(Dato result)` se usarán para crear y cerrar un `ProgressDialog` con estilo Spinner que reflejará el proceso de almacenamiento.

4. Pruebas

Mientras se ha desarrollado los módulos de este trabajo se han realizado las pruebas que verifican la funcionalidad correctamente. Para ello se ha realizado una serie de pruebas unitarias manualmente para probar los objetivos principales de este sistema. Como el sistema se compone de varias plataformas, utilizaremos las salidas por sus respectivas consolas para mostrar el resultado de las pruebas. En estas pruebas se ha usado el Samsung Galaxy 3 mini para la aplicación Android y el navegador Firefox para la aplicación Web.

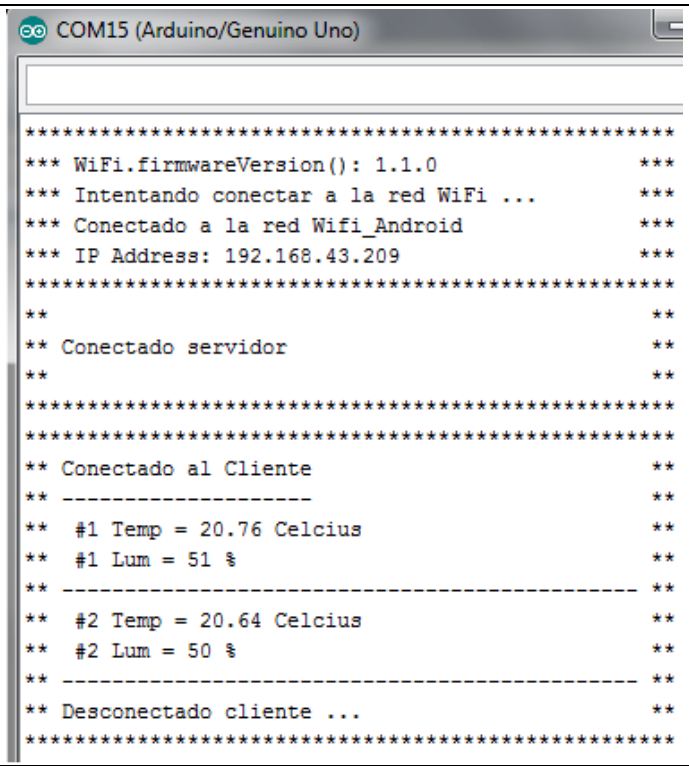
Para las pruebas relacionadas con Arduino utilizaremos el monitor serie que nos proporciona el IDE de programa Arduino. En las otras plataformas visualizaremos por las consolas que proporcionan dichas plataforma, el *LogCat* para Android y la *Console Developer Google* para la Cloud, empleando el sistema de registro. Además para testear los Endpoints, Google nos proporciona una API Explorer para realizar generar los servicios web del Backend. Accedemos a ella de la siguiente forma:

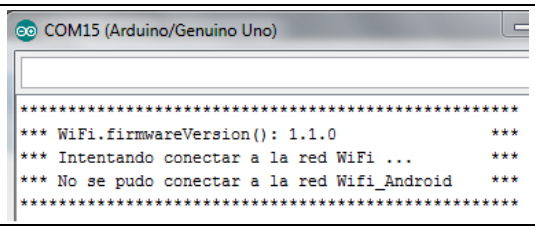
https:// cloudserver-976.appspot.com/_ah/api/explorer

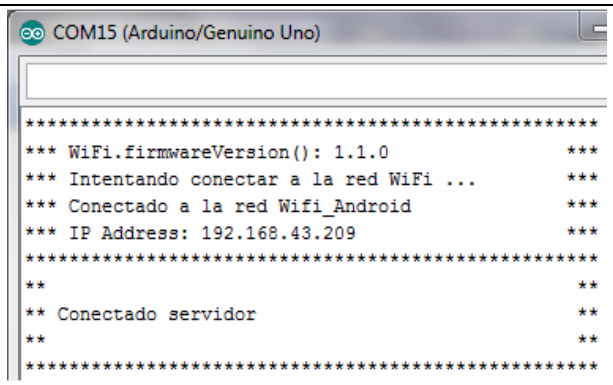
Estas pruebas se clasifican en tres grupos de la siguiente manera:

- Sistema de monitorización
  - M-001: Monitorización Valida
  - M-002: Verificación Servidor Arduino Desconectado
  - M-003: Verificación Red WiFi Desconectada
- Procesamiento de datos
  - D-001: Acceso Aplicación Android
  - D-002: Insertar Dato Monitorizado
  - D-003: Eliminar Dato
  - D-004: Filtrar Datos
- Gestión de entidades
  - E-001: Acceso Aplicación Web
  - E-002: Insertar Usuario
  - E-003: Editar Usuario
  - E-004: Activar Alerta
  - E-005: Eliminar Usuario

4.1 Sistema de monitorización

Código	M-001
Descripción	El cliente Android se conecta al servidor Arduino y este transmite los valores de temperatura y luminosidad. Y luego cierra conexión.
Salida Monitor Serie	
Salida Consola Android	CloudServer ----- Comienza Monitorización ----- CloudServer Conectando al Servidor /192.168.43.209:7000 CloudServer Conectado al servidor /192.168.43.209:7000 CloudServer ---------- CloudServer Cliente.#1 Temperatura = 20.76 °C CloudServer Cliente.#1 Luminosidad = 51 % CloudServer ---------- CloudServer Cliente.#2 Temperatura = 20.64 °C CloudServer Cliente.#2 Luminosidad = 50 % CloudServer ---------- CloudServer Cliente Desconectado CloudServer ----------

Código	M-002
Descripción	El cliente Android intenta conectarse al servidor Arduino, el cual no está conectado a la red WiFi.
Salida Monitor Serie	
Salida Consola Android	CloudServer ----- Comienza Monitorización ----- CloudServer Conectando al Servidor /192.168.43.209:7000 CloudServer ConnectException: Conexion fallida al Servidor CloudServer ----------

Código	M-003
Descripción	El servidor Arduino está conectado y el cliente Android no está conectado a la red WiFi.
Salida Monitor Serie	
Salida Consola Android	CloudServer ----- Comienza Monitorización ----- CloudServer Red Wifi Desconectada CloudServer ----------

4.2 Procesamiento de datos

<table>
<tr><td>Código</td><td>D-001</td></tr>
<tr><td>Descripción</td><td>Un usuario accede al sistema, mediante la aplicación Android.</td></tr>
<tr><td>Salida Consola Android</td><td>
<pre>
CloudServer   -------- Comienza Acceso -----------
CloudServer   Datos Introducidos:
CloudServer   Email = usuario@test.com
CloudServer   Contraseña = 1234
CloudServer   Verificando Datos ...
CloudServer   Usuario = {"email":"usuario@test.com","nombre":"usuario","password" ⏎
                 :"1234","rol":"Usuario","kind":"usuarioendpoint#resourcesItem","eta ⏎
                 g":"\"PxrkNiu8ZDVYEIV-NEiAx8kEIDU/sxUH4ywptT7ZWCRRJ-SjKvnGvTk\""}
CloudServer   ------------------------------------
</pre>
</td></tr>
<tr><td colspan="2">Salida Consola Google</td></tr>
<tr><td colspan="2">
<pre>
02:53:40.823 200   148 B   74 ms /_ah/spi/app.cloud.endpoints.UsuarioEndpoint.loginUsuario

79.148.251.164 - - [17/Nov/2015:17:53:40 -0800] "POST /_ah/spi/app.cloud.endpoints.UsuarioEndpoint.loginUsuario HTTP/1.1" 200 148 - "Google-HTTP-Java-Client/1.18.0-rc (gzip)" "cloudserver-976.appspot.com" ms=74 cpu_ms=42 cpm_usd=1.654e-05 instance=00c61b117c0074cc4240d80a4df8c72e0a3a4636 app_engine_release=1.9.29 trace_id=53e29cee054cecc9a170708e9e425e39

02:53:40.878 app.cloud.endpoints.UsuarioEndpoint loginUsuario: Verificando Email usuario@test.com y Contraseña 1234 ...

02:53:40.895 app.cloud.endpoints.UsuarioEndpoint loginUsuario: Datos verificados

02:53:40.895 app.cloud.endpoints.UsuarioEndpoint loginUsuario: Usuario = Usuario[usuario@test.com, usuario, 1234, Usuario, {[]},{[]}]
</pre>
</td></tr>
<tr><td>API Explorer</td><td>
Request
<pre>
GET https://1-dot-cloudserver-976.appspot.com/_ah/api/usuarioendpoint/v1/loginUsuario?email=usuario%40test.com&password=1234
</pre>
Response
<pre>
200 OK

- Show headers -

-{
  "email": "usuario@test.com",
  "nombre": "usuario",
  "password": "1234",
  "rol": "Usuario",
  "kind": "usuarioendpoint#resourcesItem",
  "etag": "\"PxrkNiu8ZDVYEIV-NEiAx8kEIDU/sxUH4ywptT7ZWCRRJ-SjKvnGvTk\""
}
</pre>
</td></tr>
</table>

Código	D-002
Descripción	El dato monitorizado, en este caso de temperatura, se almacena en la Cloud.
Salida Consola Android	CloudServer ----- Guardar Dato Monitorizado ---- CloudServer Dato Monitorizado: CloudServer tipo = Temperatura CloudServer valor = 21 CloudServer fecha = 2015-11-18T04:45:58.721+01:00 CloudServer Insertado Dato CloudServer usuario = {"datos":[{"appId":"s~cloudserver-976","complete":true,"id":"5108667640709120","kind":"Dato"}],"email":"usuario@test.com","nombre":"usuario","password":"1234","rol":"Usuario","kind":"usuarioendpoint#resourcesItem","etag":"\"PxrkNiu8ZDVYEIV-NEiAx8kEIDU/b6Rte-O3f1-CTs9zPDcETh4nMac\""} CloudServer ----- Guardar Dato Monitorizado ----

Salida Consola Google

```
03:46:04.070 200 257 B 143 ms /_ah/spi/app.cloud.endpoints.DatoEndpoint.insertDato

79.148.251.164 - - [17/Nov/2015:18:46:04 -0800] "POST /_ah/spi/app.cloud.endpoints.DatoEndpoint.insertDato HTTP/1.1" 200 257 - "Google-HTTP-Java-Client/1.18.0-rc (gzip)" "cloudserver-976.appspot.com" ms=143 cpu_ms=138 cpm_usd=2.8720999999999998e-05 instance=00c61b117c80baa713928ebb69f55f381ffcfd10 app_engine_release=1.9.29 trace_id=c2c35e5ba99c5756acdf07e1f99b7ff5

03:46:04.203 app.cloud.endpoints.DatoEndpoint insertDato: Dato = Dato[Dato(5108667640709120), 21, Wed Nov 18 03:45:58 UTC 2015, {[usuario@test.com]}]

03:46:05.091 200 243 B 147 ms /_ah/spi/app.cloud.endpoints.UsuarioEndpoint.asignarDato

79.148.251.164 - - [17/Nov/2015:18:46:05 -0800] "POST /_ah/spi/app.cloud.endpoints.UsuarioEndpoint.asignarDato HTTP/1.1" 200 243 - "Google-HTTP-Java-Client/1.18.0-rc (gzip)" "cloudserver-976.appspot.com" ms=147 cpu_ms=80 cpm_usd=2.7157e-05 instance=00c61b117c80baa713928ebb69f55f381ffcfd10 app_engine_release=1.9.29 trace_id=874bf37628d65795a4082374ed8c0e98

03:46:05.148 app.cloud.endpoints.UsuarioEndpoint asignarDato: Asignando al Usuario(usuario@test.com) el Dato(5108667640709120)

03:46:05.232 app.cloud.endpoints.UsuarioEndpoint asignarDato: Usuario = Usuario[usuario@test.com, usuario, 1234, Usuario, {[Dato(5108667640709120)]
```

Código	D-003
Descripción	Un usuario elimina un dato, en este caso de Temperatura, desde la aplicación Android.
Salida Consola Android	CloudServer -------- Eliminacion Dato ---------- CloudServer Eliminar Dato(5107102930436096) del Usuario(usuario@test.com) CloudServer Datos asignados = 2 CloudServer Dato eliminado CloudServer Datos asignados = 1 CloudServer ---------- CloudServer DatoEndpointsTask(): params[0].tipo = Temperatura CloudServer DatoEndpointsTask(): params[1].usuario = usuario@test.com

Salida Consola Google

Código	D-004
Descripción	Filtrar un listado de datos entre un rango de valores, en este caso Luminosidad.
Salida Consola Android	CloudServer --------- Listado de Datos ------------- CloudServer Datos asignados = [{"appId":"s~cloudserver-976","complete":true,"id":"5690665774088192","kind":"Dato"}, {"appId":"s~cloudserver-976","complete":true,"id":"5646748928180224","kind":"Dato"}] CloudServer -------- Filtrado por Valores ---------- CloudServer Valor Minimo = 40 CloudServer Valor Maximo = 60 CloudServer Datos Filtrados = [{"fecha":"2015-11-18T19:26:04.369Z","id":{"appId":"s~cloudserver-976","complete":true,"id":"5690665774088192","kind":"Dato"},"tipo":"Luminosidad","usuarios":["usuario@test.com"],"valor":51,"kind":"datoendpoint#resourcesItem"}] CloudServer ----------

Salida Consola Google

API Explorer	Request
	
	GET https://1-dot-cloudserver-976.appspot.com/_ah/api/datoendpoint/v1/filterDataByValue?email=usuario%40test.com&tipo=Luminosidad&valorMaximo=60&valorMinimo=40
	
	Response
	
	200 OK
	
	- Show headers -
	
	{
	"items": [
	{
	"id": {
	"kind": "Dato",
	"appId": "s~cloudserver-976",
	"id": "5690665774088192",
	"complete": true
	},
	"valor": 51,
	"fecha": "2015-11-18T19:26:04.369Z",
	"tipo": "Luminosidad",
	"usuarios": [
	"usuario@test.com"
	],
	"kind": "datoendpoint#resourcesItem"
	}
	],
	"kind": "datoendpoint#resources",
	"etag": "\"ut6wb3Gwld2yF47ZYtf1Lj5hN4A/XmoSXN4ddoEPFuCD-D9bwKpVWEk\""
	}

4.3 Gestión de entidades

Código	E-001
Descripción	Un usuario accede al sistema, mediante la aplicación web.
Salida Consola Google	

```
17:45:38.192 200     2,6 KB    9,45 s  /loginServlet

79.148.251.164 - - [18/Nov/2015:08:45:38 -0800] "POST /loginServlet HTTP/1.1" 200 2666 http://1-dot-cloudserver-976.appspot.com/ "Mozilla/5.0 (Windows NT 6.1; WOW64; rv:42.0) Gecko/20100101 Firefox/42.0" "1-dot-cloudserver-976.appspot.com" ms=9447 cpu_ms=5912 cpm_usd=0.000297948 instance=00c61b117c2fd40b1f881ca2de5a9be423559544 app_engine_release=1.9.29 trace_id=6619af45e9ac614884d6f2e2b0161975

17:45:38.260 app.cloud.servlet.LoginServlet doPost: Email = usuario@test.com
17:45:38.260 app.cloud.servlet.LoginServlet doPost: Contraseña = 1234
17:45:41.933 app.cloud.dao.AbstractFacade find: Verificando datos ...
17:45:43.813 app.cloud.servlet.LoginServlet doPost: Datos Verificados
17:45:43.815 app.cloud.servlet.MenuServlet doPost: Usuario = Usuario[usuario@test.com, usuario, 1234, Usuario, {[Dato(5684049913839616), Dato(4857926107791360)]},{[]}]
```

Código	E-002
Descripción	Un administrador inserta un usuario en el sistema desde la aplicación web.

Salida Consola Google

```
18:27:28.624 200    1,84 KB    195 ms /insertarUsuarioServlet
79.148.251.164 - - [18/Nov/2015:09:27:28 -0800] "POST /insertarUsuarioServlet HTTP/1.1" 200 1882 http://1-dot-cloudserver-976.appspot.com/insertarUsuarioServlet "Mozilla/5.0 (Windows NT 6.1; WOW64; rv:42.0) Gecko/20100101 Firefox/42.0" "1-dot-cloudserver-976.appspot.com" ms=195 cpu_ms=129 cpm_usd=0.000210329 instance=00c61b117c1333110b4cd6ad15e334ebef2d2c71 app_engine_release=1.9.29 trace_id=b86c2924ba9980eb6c022d062d0611d7
18:27:28.687 app.cloud.servlet.InsertarUsuarioServlet doPost: Email = UsuarioNuevo@test.com
18:27:28.687 app.cloud.servlet.InsertarUsuarioServlet doPost: Nombre = Usuario Nuevo
18:27:28.687 app.cloud.servlet.InsertarUsuarioServlet doPost: Contraseña = 1234
18:27:28.687 app.cloud.servlet.InsertarUsuarioServlet doPost: Rol = Usuario
18:27:28.705 app.cloud.servlet.InsertarUsuarioServlet doPost: Usuarios del sistema = 4
18:27:28.705 app.cloud.dao.AbstractFacade find: Verificando datos ...
18:27:28.715 app.cloud.dao.AbstractFacade find: Usuario no encontrado con id UsuarioNuevo@test.com
18:27:28.803 app.cloud.servlet.InsertarUsuarioServlet doPost: Usuario Insertado = Usuario[UsuarioNuevo@test.com, Usuario Nuevo, 1234, Usuario, {null},{null}]
18:27:28.818 app.cloud.servlet.InsertarUsuarioServlet doPost: Usuarios del sistema = 5
```

Código	E-003
Descripción	Un usuario edita la información de su perfil en el sistema desde la aplicación web.

Salida Consola Google

```
18:47:02.248 200    1,82 KB    264 ms /editarUsuarioServlet
79.148.251.164 - - [18/Nov/2015:09:47:02 -0800] "POST /editarUsuarioServlet HTTP/1.1" 200 1867 http://1-dot-cloudserver-976.appspot.com/editarUsuarioServlet "Mozilla/5.0 (Windows NT 6.1; WOW64; rv:42.0) Gecko/20100101 Firefox/42.0" "1-dot-cloudserver-976.appspot.com" ms=264 cpu_ms=54 cpm_usd=0.000208653 instance=00c61b117c47f7559f7bfb59d8d864f56980e6d5 app_engine_release=1.9.29 trace_id=910721f873b52524a53b90d7f98a27b9
18:47:02.310 app.cloud.servlet.EditarUsuarioServlet doPost:  Email = UsuarioNuevo@test.com
18:47:02.310 app.cloud.servlet.EditarUsuarioServlet doPost:  Nombre Modificado = NombreModificado
18:47:02.355 app.cloud.dao.AbstractFacade edit: Editando entidad ...
18:47:02.397 app.cloud.servlet.EditarUsuarioServlet doPost: Usuario Modificado = Usuario[UsuarioNuevo@test.com, NombreModificado, 1234, Usuario, {[]},{[]}]
```

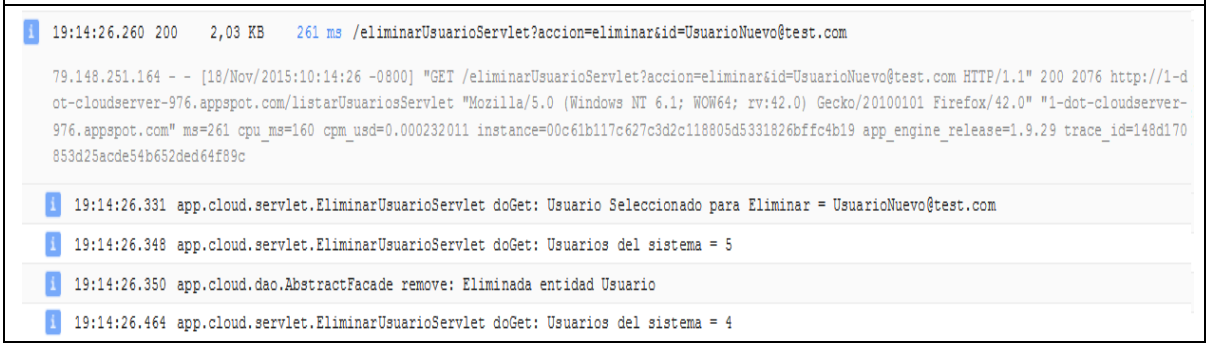
Código	E-004
Descripción	Un usuario activa una alerta en el listado de sus alertas desde la aplicación web.

Salida Consola Google

```
18:57:23.727 200    2,08 KB    166 ms /listarAlertasServlet?accion=Activar&id=5759778777202688
79.148.251.164 - - [18/Nov/2015:09:57:23 -0800] "GET /listarAlertasServlet?accion=Activar&id=5759778777202688 HTTP/1.1" 200 2129 http://1-dot-cloudserver-976.appspot.com/listarAlertasServlet?accion=Desactivar&id=5759778777202688 "Mozilla/5.0 (Windows NT 6.1; WOW64; rv:42.0) Gecko/20100101 Firefox/42.0" "1-dot-cloudserver-976.appspot.com" ms=166 cpu_ms=55 cpm_usd=0.000237934 instance=00c61b117c6e1705e5422226fabf7d8c266e31e5 app_engine_release=1.9.29 trace_id=7d5fb3ffaac4aec71031363203e3aa6a
18:57:23.801 app.cloud.servlet.ListarAlertasServlet doPost: Alerta = Alerta[Alerta(5759778777202688), Temperatura, 35, 45, false]
18:57:23.836 app.cloud.dao.AbstractFacade edit: Actualizando entidad ...
18:57:23.836 app.cloud.servlet.ListarAlertasServlet doPost: Alerta Activada = Alerta[Alerta(5759778777202688), Temperatura, 35, 45, true]
```

Código	E-005
Descripción	Un administrador elimina un usuario del sistema desde la aplicación web.

Salida Consola Google



5. Conclusiones y Extensiones

Actualmente, el uso de las tecnologías software y hardware para crear necesidades en la sociedad está muy desarrollado en este ámbito. Por tanto, el objetivo principal planteado para este trabajo era integrar tres tecnologías como Arduino, Android y la Cloud en un mismo sistema, cuyo propósito era monitorizar una serie de sensores.

Dicho objetivo se ha alcanzado mediante un sistema de monitorización en tiempo real de dos sensores conectado al Arduino, en nuestro trabajo de temperatura y de luminosidad, donde se requería una comunicación entre las plataformas. De esta manera, podemos visualizar los valores a través de un dispositivo móvil para su inmediato almacenamiento en la nube.

Otro de los objetivos era la creación de una aplicación móvil en Android y una aplicación Web lo más intuitiva posible para tratar la información del sistema.

Además de estos objetivos, se ha conseguido profundizar en una tecnología no desarrollado durante el periodo de estudio, como es la Cloud. Este objetivo se ha logrado satisfactoriamente en este trabajo.

Este sistema puede ser usado en diferentes sectores o entornos, como son pruebas científicas de un laboratorio o la automatización de una vivienda. En estos casos, se proporciona diferente funcionalidades útiles como son las alertas, por ello, este sistema da la oportunidad de configurar alertas.

Este trabajo se ha diseñado para que sea fácilmente ampliable y mejorable en varios aspectos, por eso planteamos las siguientes mejoras al sistema:

- Ampliar el modelo para aceptar más tipos de sensores para monitorizar, crear más funcionalidades.
- Utilizar más servicios ofrecido por la Cloud, por ejemplo, como enviar notificaciones al correo electrónico para informar de las alertas.
- Añadir más seguridad en las API de las aplicaciones usando el protocolo OAuth proporcionado por Google.
- Establecer conexión entre el servidor del Arduino y el servidor alojado en la nube de una forma directa. Además de poder monitorizar datos en segundo plano desde la aplicación Android.

6. Bibliografía

1. **Alfaomega Grupo Editor, S.A**. *Arduino. Curso práctico de formación.* Torrente, Óscar. México, Febrero 2013
2. **Arduino**. [En linea] Octubre 2015. https://es.wikipedia.org/wiki/Arduino
3. **Arduino CC, Learning**. [En linea] Octubre 2015. https://www.arduino.cc/en/Guide/Introduction
4. **Arduino CC, Home.** *Upgrading the Wifi shield firmware***.** [En línea} Octubre 2015. https://www.arduino.cc/en/Hacking/WiFiShieldFirmwareUpgrading
5. **Bricogeek. Arduino WiFi Shield SD**. [En linea] Septiembre 2015 http://tienda.bricogeek.com/home/446-arduino-wifi-shield-sd.html
6. **Cooking-hacks**. *DataSheet thermistor NTC 159-282-86001*. [En linea] Abril 2015 http://www.cooking-hacks.com/skin/frontend/default/cooking/pdf/159-282-86001.pdf
7. **Fotorrsitor**. [En linea] Abril 2015. https://es.wikipedia.org/wiki/Fotorresistor
8. **Arduino CC. Download**. *Arduino Software 1.6.5*. [En linea] Abril 2015 https://www.arduino.cc/en/Main/Software
9. **Android**. [En linea] Octubre 2015 https://es.wikipedia.org/wiki/Android
10. **Google, Cloud Platform**. [En linea] Mayo 2015. https://cloud.google.com/
11. **Curso Cloud Computing, Parte 2**. *Google App Engine*. [En linea] Abril 2015. http://es.slideshare.net/dipina/curso-cloud-computing-parte-2-google-app-engine
12. **Google Developers. Google Plugin for Eclipse.** *Getting Started*. [En linea] Abril 2015. https://developers.google.com/eclipse/docs/getting_started
13. **Google, App Engine.** *Quotas.* [En linea] Abril 2015. https://cloud.google.com/appengine/docs/quotas
14. **Google. DataStore**. *Java Datasotre API*. [En linea] Abril 2015. https://cloud.google.com/appengine/docs/java/datastore/
15. **Google, Google Cloud Enpoints**. [En linea] Abril 2015 https://cloud.google.com/appengine/docs/java/endpoints/
16. **UMA**. *Apuntes de Ingeniería del Software*. 2013.
17. **MagicDraw UML Personal Edition, 2012.** http://www.nomagic.com/products/magicdraw.html
18. **Fritzing**. [En linea] Octubre 2015. http://fritzing.org/home/
19. **Arduino CC. Learning**. *Libraries*. [En linea] Septiembre 2015. https://www.arduino.cc/en/Reference/Libraries
20. **Android, Desarrollar.** *Guias de la API*. [En linea] Junio 2015. http://developer.android.com/intl/es/guide/index.html
21. **Android, Desarrollar**. *Referencia*. [En linea] Junio 2015. http://developer.android.com/intl/es/reference/packages.html
22. **Data Access Object**. **DAO**. [En linea] Abril 2015. https://es.wikipedia.org/wiki/Data_Access_Object
23. **Patrón de diseño**. *Facade*. [En linea] Abril 2015. https://es.wikipedia.org/wiki/Facade_%28patr%C3%B3n_de_dise%C3%B1o%29

24. **UMA**. *Apuntes de Tecnología en Aplicaciones Web.* 2013.
25. **UMA**. *Apuntes de Software para Sistemas Empotrados y Dispositivos Móviles*. 2014.
26. **Google Developers. Charts**. [En linea] Octubre 2015. https://developers.google.com/chart/interactive/docs/quick_start

7. Anexos

7.1 Manual de usuario

Este sistema contiene dos aplicaciones para interactuar con el usuario. De manera que este capítulo servirá de ayuda y guía para los nuevos usuarios del sistema.

7.1.1 Aplicación Android

Lo primero que debemos realizar es instalar la aplicación en un dispositivo móvil Android. Para ello tenemos que ejecutar el instalador *CloudServer.apk*, ubicado en el siguiente directorio "*CloudServer\bin*". Para que te permita instalar la aplicación debemos marcar la opción *Fuentes desconocida* en *"Ajustes>Seguridad>Administración de dispositivo"*.

Una vez instalada, pulsaremos este icono  en el escritorio para entrar a ella. Se abrirá la pantalla para introducir tu email y contraseña con la que estas registrado en el sistema, y un botón ***"Entrar"*** para acceder al sistema. En este proceso aparecerá un cuadro de espera mientras verificas tus datos.

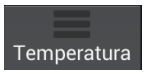
Cuando accedemos no aparecerá la pantalla principal de la aplicación donde podemos visualizar los siguientes elementos:

- 

: El nombre del usuario conectado.
- 

: Los valores donde se mostrará la temperatura (grados Celsius) y la luminosidad (porcentaje).
- 

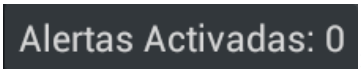
/ 

: Este botón se utilizar para activar o desactivar la monitorización de datos. En el caso de que estuviera conectado y se pulsará empezará el proceso de almacenamiento de datos monitorizados.
- 

/ 

: Estos botones llevarán a un listado de datos del tipo cuando es pulsado.
- 

: Este texto indica la última vez que se activo la monitorización
- 

/ 

: Este espacio se utilizará para informar al usuario de la conexión, o para mostrar texto con tiempo limitado.

- 

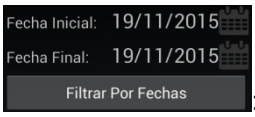
/ 

: Este botón solo estará visible si el usuario conectado es un Administrador. Cuando se pulsa llevará a un listado del historial completo de datos y los usuarios registrado en el sistema.

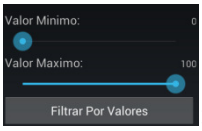
- 

: Este botón cerrará la sesión del usuario conectado y volverá aparecer la pantalla del login. Está situado en la ActionBar.

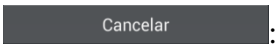
Desde las pantallas donde se muestra listados de datos podremos realizar una acción para filtrar los datos, pulsando el botón de menú del dispositivo, a la opción ***"Filtrar".*** Posterior a esta acción aparecerá una pantalla nueva donde tendrá los siguientes elementos:

- 

: En esta parte del filtrado se configurará las fechas de inicio y final pulsando el texto donde se muestra la fecha actual. Cuando se pulsa en el texto saldrá un cuadro para cambiar el dia, mes y año. Si pulsamos el botón **"*Filtrar Por Fecha*"** volveremos al listado con dicha acción.

- 

: En esta parte del filtrado se configurará el rango de valores para filtrar con la ayuda de las barras deslizantes. Si pulsamos el botón **"*Filtrar Por Valores*"** volveremos al listado con dicha acción.

- 

: Este botón nos devolverá al listado sin realizar filtros.

7.1.2 Aplicación Web

Para acceder a la aplicación web del sistema, tendremos que introducir esta dirección:

http://cloudserver-976.appspot.com/

Cuando accedemos a dicha dirección aparecerá la ventana de Login para introducir el email y contraseña del usuario para acceder al sistema. En esta venta también existe la posibilidad de ir a la ventana de registro de nuevos usuarios. En dicha ventana se introducirá el email, nombre, y contraseña con las cuales quiere ser registrado.

Una vez que el usuario entra en el sistema mostrará la ventana del menú principal donde aparecerá, en el cuadro principal de esta ventana, dos gráficos para cada tipo de datos que representa los valores medio de datos monitorizados en los últimos siete días. También aparecerá un cuadro con los datos monitorizados por el usuario en los últimos sietes días de cada tipo y la media de estos datos.

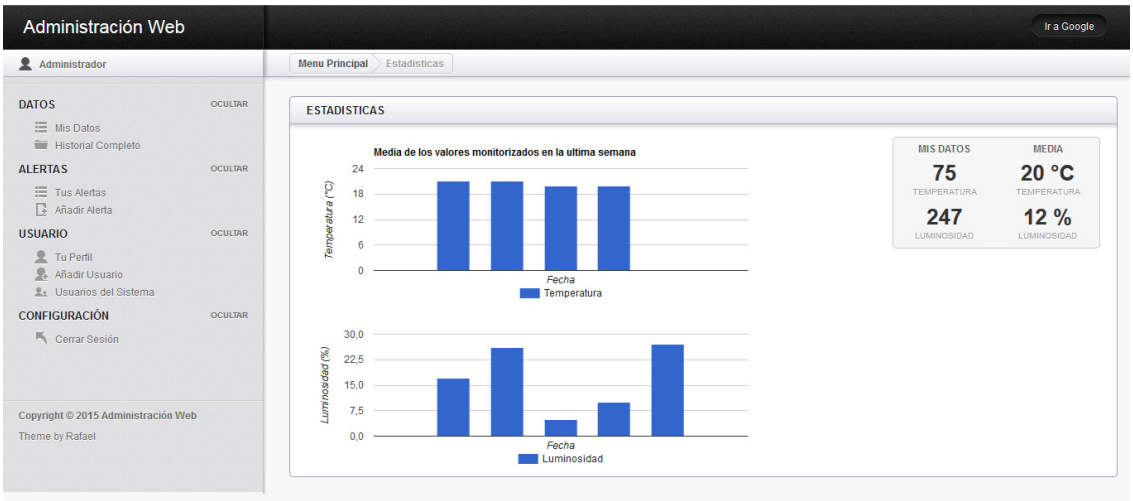


Figura 7.1: Ventana del Menú Principal (Web)

Como vemos en la imagen, tendremos una barra lateral situada en la izquierda con las acciones que pueden llevarse a cabo, en este caso es la vista como Administrador. Para el caso del Usuario no se visualizará las opciones de "***Historial Completo***", ***"Añadir Usuario"*** y ***"Usuarios del sistema"***.

En los listados de esta aplicación existirá acciones adicionales junto a los elementos dependiendo la lista a mostrar. Estos acciones son:

-  : Este botón llevara a la ventana de edición del correspondiente elemento seleccionado, como usuario o alerta.
-  : Este botón eliminará el elemento seleccionado del sistema si es un dato se eliminará del usuario conectado o del sistema si la acción es ejecutada por un administrador.
- 

: Este botón cambiará el estado de la alerta. Aparecerá uno u otro según el estado que este dicha alerta.

En el caso particular del listado de datos aparecerá a la derecha de la ventana un cuadro para introducir fechas y valores para realizar el filtrado de datos, como vemos a continuación.

Figura 7.2: Cuadro de filtrado (Web)