Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Control y Monitorización de hábitats para animales mediante Raspberry-Pi, Fiware, Spring y Android

Autor: Leopoldo Angulo Gallego
Tutor: María Teresa Ariza Gómez

Dpto. de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2019

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Control y Monitorización de hábitats para animales mediante Raspberry-Pi, Fiware, Spring y Android

Autor:
Leopoldo Angulo Gallego

Tutor:
María Teresa Ariza Gómez
Profesor titular

Dpto. de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2019

Proyecto Fin de Carrera: Control y Monitorización de hábitats para animales mediante Raspberry-Pi, Fiware, Spring y Android

Autor: Leopoldo Angulo Gallego

Tutor: María Teresa Ariza Gómez

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2019

El Secretario del Tribunal

A mi familia

A mis amigos

A mis maestros

Agradecimientos

Antes de comenzar con el contenido de la memoria de este Trabajo Fin de Grado, me gustaría dejar constancia y expresar mis agradecimientos, ya que no es un Trabajo más, sino que es un punto y final de una gran etapa. Etapa cargada de "sudor y lágrimas", que llega a su fin con la obtención del Título de Graduado en Ingeniería de las Tecnologías de Telecomunicación. Llegados a este punto, me siento en la obligación de revelar que no habría podido conseguirlo por mí solo, razón por la que dedicaré unas pocas líneas de esta Memoria en nombrar a algunas de las personas que lo hicieron posible.

En primer lugar, a esos profesores que son capaces de hacer fácil lo difícil. En particular, a María Teresa Ariza Gómez, tanto por darme la oportunidad de usar mi idea para este trabajo, como por su indudable profesionalidad que muestra diariamente como docente de sus asignaturas y en su labor de Tutora de Trabajos Fin de Grado.

En segundo lugar, a mi familia, apoyo incondicional que siempre espero tener. Agradecerles la ayuda en los momentos difíciles y las celebraciones por cada pequeña victoria obtenida. Especial agradecimiento a mis padres, por su confianza e innumerables "inversiones" en mí; y a mis abuelos, por la incansable motivación que siempre he recibido.

Otra parte que debe tener su hueco en este documento son los amigos, aquéllos que he conocido durante estos cuatro años y aquéllos que han estado ahí desde el principio. Gracias por dejarme formar parte del equipo. Equipo que siempre permanecerá unido.

Por último, a mi compañera de vida, que ha vivido conmigo todo los tropiezos y buenos momentos desde el principio y que sé, que seguirá en todas las etapas que están por venir.

Leopoldo Angulo Gallego

Sevilla, 2019

Resumen

El auge de los dispositivos conectados a Internet conduce a la búsqueda de soluciones basadas en el *Internet of Things*. Este proyecto no es más que el intento de mejorar un raro ámbito comercial con el uso de herramientas software y sistemas electrónicos como sensores, microprocesadores, plataformas de servicios inteligentes, API REST y App-Android.

El proyecto que se detallará a continuación está enfocado a monitorizar y controlar hábitats de animales como terrarios y acuarios, los que necesitan unas condiciones específicas para la correcta vida de estos. Se ha enfocado en la parte del Gestor de una o varias tiendas de animales que contienen multitud de datos a controlar.

Para facilitar esa tarea, se ha centrado el estudio en un número reducido de hábitats y en una sola tienda. Pero es posible extender las herramientas desarrolladas a entornos reales, donde un Gestor está encargado de varias tiendas con diversos Terrarios/Acuarios a su disposición.

Se ha usado una Raspberry-Pi como agente IoT, a la que van conectados los sensores. Este dispositivo ejecuta un programa encargado de Transmitir los datos al Context-Broker de Fiware. Con herramientas como Cygnus y MySQL se almacenan los datos en una BBDD que es accedida por un Servidor Spring usando Hibernate y JPA.

El Gestor puede monitorizar estos datos a través de una aplicación móvil Android que se comunica con el servidor usando una API-REST. También podrá actuar sobre algún dispositivo y recibir notificaciones.

Abstract

The rise of Internet devices leads to the search for solutions based on the Internet of Things. This project is only an attempt to improve a rare commercial environment with the use of software tools and electronic systems such as sensors, microprocessors, IoT platforms, Rest-APIs and Android Applications.

The project is focused on monitoring and controlling animal habitats such as terrariums and aquariums, which need specific conditions for their healthy life. It has focused exactly on the part of the Manager of pet stores which contain a multitude of data to control them.

To do easier the task, the study has focused on a single store with a small number of habitats. But it is possible to extend the tools developed to real environments. Where the Manager is in charge of several stores with a lot of terrariums/aquariums at his disposal.

A Raspberry-Pi has been used as an IoT Agent, to which the sensors are connected. This device runs a program for transmitting data to the Context Broker. Using tools such as Cygnus and MySQL we store the data in a database that is accessed by a Spring Server using Hibernate ad JPA.

The Manager can monitor this data through an android application that communicates with the server using a Rest-API. He can also act on some device and receive notifications.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

BBDD	Base de Datos
IoT	Internet de las Cosas
STS	Spring Tool Suite
CB	Context Broker
Raspberry	Raspberry Pi Model B+
BNC	Conector *Bayonet Neill-Concelman*
MV	Máquina Virtual
ETSI	Escuela Técnica Superior de Ingeniería
API	Interfaz de Programación de Aplicaciones (Application Programming Interface)
GPIO	Entrada/Salida de Propósito General (General Purpose Input/Output)
ADC	Conversor Analógico Digital
I2C	Inter Circuitos Integrados (Inter-Integrate Circuit)
UI	User Interface
APT	Herramienta Avanzada de Empaquetado (Advance Packagin Tool)

1 INTRODUCCIÓN

Intelligence is the ability to adapt to change.

- Stephen Hawking -

1.1 Motivación

La principal motivación para la realización de este proyecto ha sido la de simplificar las tareas de un Gestor[1] de una tienda, especialmente en una tienda de animales, a través de la monitorización y control de los hábitats donde viven éstos. Para que se pueda llevar a cabo esta tarea, se han utilizado diversas tecnologías para dar soporte a toda la plataforma, de manera que permita controlar remotamente desde una aplicación Android los parámetros básicos de los terrarios[2] y acuarios[3]. Por ejemplo, la temperatura, la humedad, el PH del agua …

Debido al auge del Internet de las Cosas (IoT), se ha decidido basar nuestro proyecto en tecnologías como el Context Broker de Fiware. Además de ésta se han usado otras herramientas tecnológicas, que se comentarán en apartados posteriores, de forma que permita la implementación de un sistema que va desde la instalación de los sensores hasta el desarrollo de la aplicación de usuario.

Teniendo en cuenta que los Smartphone son los dispositivos más usados actualmente, se ha desarrollado en Android la aplicación final que utilizará el usuario.

Destacar también como motivación, la de indagar y profundizar en estas tecnologías de gran interés, y que son muy demandadas actualmente por entidades y empresas.

1.2 Objetivos

El objetivo del Trabajo es la realización de un sistema formado por un Agente IoT conectado a los diferentes sensores, que este sistema envíe la información al Context Broker de Fiware y que los almacenen en una base de datos. Por otra parte, se implementa un Servidor Web Spring que pueda acceder a los datos de la BBDD y que esté accesible a través de la aplicación Android.

Por tanto, el objetivo de este proyecto se extiende más allá del simple monitoreo de sensores para controlar hábitats de animales, ya que éste engloba el estudio de diversas herramientas y tecnologías funcionalmente distintas y gratuitas para obtener y maximizar un sistema de control-monitorización remoto. En nuestro caso particular es usado para controlar tiendas de animales, pero es fácilmente aplicable y extensible para el

[1] Gestor: en este documento se considerada como Gestor al encargado del cuidado de todos los hábitats de una o varias tiendas.
[2] Terrario: recipiente donde se reproducen fielmente las condiciones ambientales necesarias para seres vivos terrestres.
[3] Acuarios: recipiente donde se reproducen fielmente las condiciones ambientales necesarias para seres vivos acuáticos.

mantenimiento y la gestión de multitud de establecimientos.

1.3 Antecedentes

Tener a familiares de mi entorno apasionados por los reptiles y peces exóticos me ha permitido conocer las dificultades de la cría y del mantenimiento de éstos. Lo que me condujo a ingeniar una solución de monitorización y de control para facilitarles esta tarea.

En la asignatura Sistemas Operativos de segundo curso del Grado, implementamos una posible solución a algunos de los principales problemas de los hábitats. En este trabajo se monitorizaba un sensor de temperatura y humedad conectado a una Raspberrry Pi mediante las tecnologías dadas durante el curso.

Este proyecto extiende la idea de monitorización de ese trabajo inicial y utiliza tecnologías de IoT, Servicios Web, Android… para llevar a cabo una versión completa y funcional.

1.4 Descripción de la solución

En este apartado se va a comentar, de forma general, cómo se ha implementado la solución para alcanzar los objetivos anteriormente expuestos.

1.4.1 Objetivos específicos

- Obtención de datos: los datos se obtienen a través de dos sensores distintos. El primero es un DHT-11, que se encarga de obtener la temperatura y la humedad. El segundo es solo para acuarios y es un medidor de PH encargado de obtener el nivel de 0-14 que tiene el agua. Las especificaciones de estos sensores se detallan en el apartado *2.1 Recursos Hardware*.
- Conexión de sensores: los sensores se conectan a la Raspberry. El sensor de temperatura se conecta directamente a los pines GPIOs de ésta, mientras que el sensor encargado de medir el PH necesita de un conversor ADC para transformar la salida analógica en digital. Este conversor está conectado a la interfaz I2C[4] de la Raspberry. Todos los datos son obtenidos y almacenados mediante el programa *daemon_sensor.py* desarrollado en Python.
- Envío de datos: el programa ejecutado en la Raspberry es también el encargado de enviar los datos y de actualizar los campos del Context Broker de Fiware, el cual está alojado en un contenedor Docker en otra máquina distinta. Para enviar y recibir los datos, éste realiza peticiones HTTP con los datos en formato JSON. De esta forma, cada vez que el sensor realice una lectura se actualiza la información en tiempo real.
- Almacenamiento de datos: para el almacenamiento utilizamos otra herramienta denominada Cygnus. Ésta funciona a través de una subscripción al Context Broker y es la encargada de almacenar los datos en una BBDD MySQL. Cygnus almacena los datos en las tablas correspondientes cada vez que se crean o se actualizan entidades y/o atributos.
- Consulta de datos: el usuario final consulta los datos a través de la aplicación *controlYoursAnimals.apk*. Ésta consulta los datos almacenados en la BBDD mediante un servidor Web desarrollado con Spring. Se ha creado una API REST que permite que la aplicación acceda a dichos datos a través de peticiones HTTP. De esta forma, el usuario final solo interactúa con el Servidor Web, quedando aislado de forma transparente la conexión de los sensores y el uso del Context Broker.
- Actuador: el programa ejecutado en la Raspberry simula la conexión de una cerradura de seguridad, siendo ésta muy importante cuando se trata de animales peligrosos que solo pueden ser manipulados por personas cualificadas. Esta cerradura puede ser abierta a través de la aplicación móvil cuando el Gestor lo desee. De esta forma se evita que se abra el hábitat de un animal venenoso por alguien que

[4] I2C: bus de comunicaciones en serie.

no es apto para manipularlo.

1.4.2 Funcionalidades

Desde la aplicación final del usuario, el sistema ofrece las siguientes funcionalidades:

- Consulta del número de tiendas disponibles para gestionar.
- Consulta del número de terrarios disponibles en cada tienda.
- Consulta del número de acuarios disponibles.
- Consultar si el animal es un animal "*peligroso"*.
- Abrir cerradura de "animal peligroso".
- Consultar temperatura en tiempo real de un terrario/acuario.
- Consultar humedad en tiempo real de un terrario/acuario.
- Consultar PH de un acuario.

1.4.3 Esquema de la arquitectura

En la Figura 1 se muestra el esquema de la arquitectura del sistema, centrándose éste en las tecnologías utilizadas.

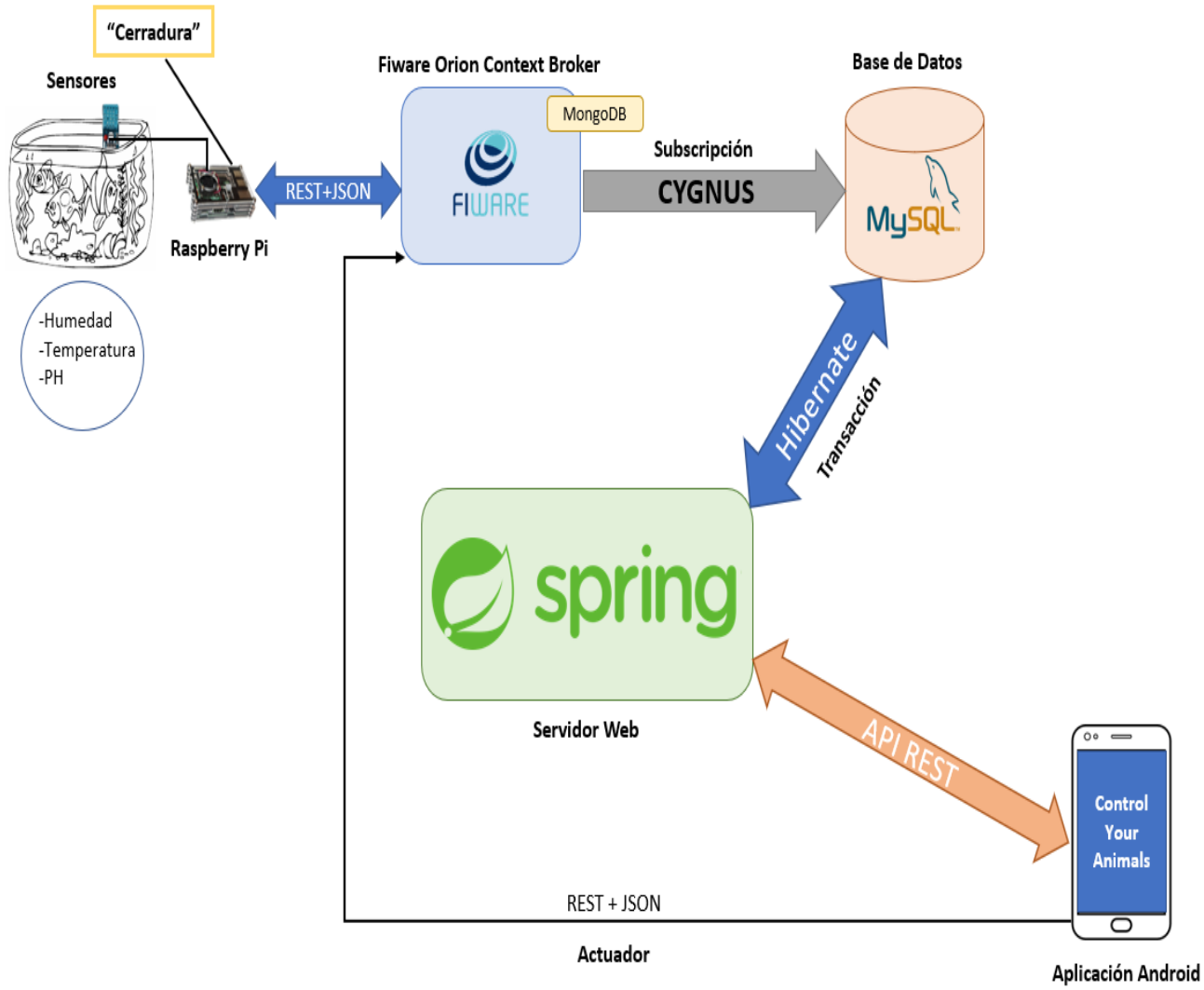

Figura 1. Esquema del Sistema.

El esquema anterior representa una visión general de la conexión de la Raspberry Pi con los sensores, así como la comunicación de los datos obtenidos con el Context Broker de Fiware. Además de esto, se utiliza la herramienta Cygnus para almacenar esta información en la Base de Datos MySQL. A esta BBDD también accede el Servidor Web mediante la herramienta Hibernate cada vez que se realiza una consulta por parte de la Aplicación Android *Control Your Animals.*

Igualmente se refleja en el diagrama que se puede actuar sobre el CB directamente con la App. Ésta permite "abrir" la cerradura de algunos de los hábitats.

1.5 Estructura de la memoria

A continuación, se proporciona un resumen de cada capítulo tratado en este Trabajo.

1. **Introducción:** motivación, objetivos y breve descripción de la solución desarrollada.
2. **Recursos utilizados**: resumen de los recursos *Software* y *Hardware* utilizados.
3. **Tecnologías utilizadas:** resumen de las tecnologías utilizada a lo largo del Proyecto.
4. **Aplicación desarrollada: Dispositivo y Context Broker:** análisis en profundidad del Dispositivo Hardware implementado, así como del Context Broker de Fiware.
5. **Aplicación desarrollada: Servidor Web y Base de Datos:** análisis del desarrollo del Servidor Web

implementado con Spring y de las consultas que éste realiza a la BBDD MySQL.

6. **Aplicación desarrollada: Aplicación en Android e Interfaz de usuario:** análisis del desarrollo de la aplicación para móviles Android con Unity.
7. **Conclusiones y líneas futuras:** problemas y futuras mejoras que han surgido a raíz del desarrollo del trabajo.

Además de los apartados anteriores se proporcionan los siguientes Anexos que servirán para la instalación y correcta ejecución de todo el sistema.

- **Anexo A:** Instalación y configuración de Context Broker con Docker.
- **Anexo B**: Instalación y configuración de Cygnus y MySQL.
- **Anexo C**: Configuración y conexión de sensores con Raspberry-Pi.
- **Anexo D:** Instalación y configuración de Unity.
- **Anexo E:** Instalación y configuración de STS.
- **Anexo F:** Configuración de la Máquina Virtual.
- **Anexo G:** Puesta en Marcha del sistema.

2 RECURSOS UTILIZADOS

640K ought to be enough for anybody.

-Bill Gates-

Se detallará a continuación los recursos utilizados para la realización del trabajo, siendo éstos los específicos para una ejecución de prueba. No necesariamente son necesarios los nombrados en los siguientes subapartados para el correcto funcionamiento del servicio, pudiéndose variar el número y la característica de alguno de éstos.

2.1 Recursos Hardware

2.1.1 Ordenador de escritorio

Para realizar el proyecto se ha utilizado un ordenador de sobremesa donde se alojará la máquina virtual con el Context Broker, la base de datos, el servidor Spring y las herramientas de pruebas y codificación. Además, se utilizará éste para el desarrollo de la Aplicación Android.

El ordenador utilizado tiene las siguientes características principales:

- Procesador: Intel(R) Core (TM) i7-7700K CPU @ 4.20GHz.
- Placa Base: MSI B350 GAMING PRO CARBON.
- Memoria RAM: 16.0GB
- Almacenamiento: 256GB SSD
- Sistema Operativo: Windows 10 Pro N.

Pudiéndose usar cualquier otro ordenador de características similares para implementar el servicio.

2.1.2 Raspberry Pi 3 Model B+

Se utiliza una Raspberry Pi como agente IoT, la cual va recabando los datos que generan los sensores. A su vez, es la responsable de enviarlos al Context Broker utilizando su interfaz Wi-Fi. En nuestro caso solo utilizaremos una Raspberry, pero, para una aplicación real del proyecto, debería haber una en cada tienda que se quiera gestionar; y tantos sensores en cada acuario/terrario como parámetros se quieran monitorizar. La que se ha utilizado se muestra en la Figura 2.

.

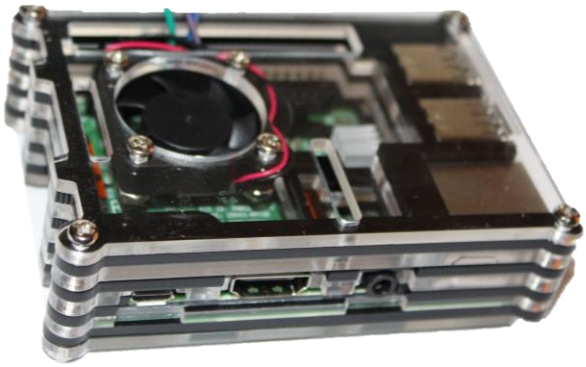

Figura 2. Raspberry Pi 3 Model B+.

Características principales:

- CPU + GPU: Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz.
- RAM: 1GB LPDDR2 SDRAM.
- Wi-Fi + Bluetooth: 2.4GHz y 5GHz IEEE 802.11.b/g/n/ac, Bluetooth.
- Ethernet: Gigabit Ethernet sobre USB 2.0 (300 Mbps)
- GPIO de 40 pines.
- HDMI.
- 4 puertos USB 2.0

2.1.3 Sensor DHT11

Sensor de humedad y temperatura que va conectado a los pines GPIO de la Raspberry. Estos sensores irán instalados en todos los hábitats de los animales, ya que la temperatura y humedad son parámetros muy importantes a tener en cuenta para la cría de animales exóticos. Este sensor se muestra en la Figura 3.

Figura 3.Sensor DHT11.

Los aspectos técnicos se encuentran en el Datasheet referenciado [1].

2.1.4 Sensor analógico PH Meter v1.1

Debido a la importancia que tiene el PH del agua para los peces en los acuarios. Se ha decidido incorporar un sensor analógico PH Meter v1.1 como el que se muestra en la Figura 4.

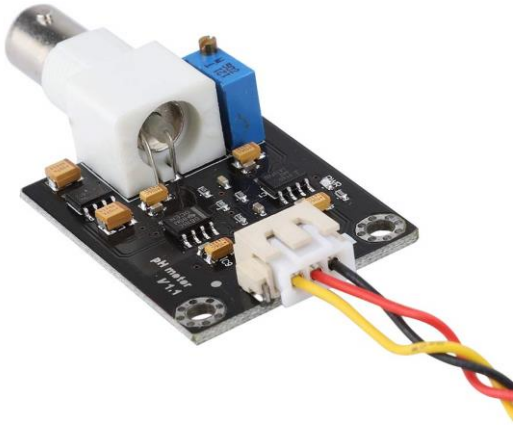

Figura 4. Sensor PH [21].

Este componente tiene las siguientes especificaciones:

- Potencia del módulo máxima: 5.00V.
- Tamaño del módulo: 43 * 32 mm.
- Rango de medición: 0 - 14 PH.
- Temperatura de medición: 0 - 60 ºC.
- Precisión: +- 0.1 pH (25 ºC).
- Tiempo de respuesta: <= 1minuto.
- Led indicador de energía.

El sensor descrito es específico para Arduino, ya que es un sensor analógico y la Raspberry Pi no dispone de pines de entrada analógicos. Para poder usar éste hemos tenido que conectarlo a un conversor ADC (MCP3008), descrito en un apartado posterior.

2.1.5 Sonda de PH de electrodo BNC

Para que el anterior componente pueda procesar las medidas necesita de una Sonda de PH conectada mediante el conector BNC. En este proyecto se ha usado una sonda como la de la Figura 5.

Figura 5. Sonda Sensor PH [21].

Esta sonda está formada por un electrodo de vidrio de PH y el electrodo de referencia de plata y cloruro de plata.

2.1.6 Microchip MCP3008

El MCP3008-I/P es un Conversor ADC de 8 Canales y 10bits. Éste es necesario para obtener la señal analógica suministrada por el medidor de PH. El conversor usado aparece en la Figura 6.

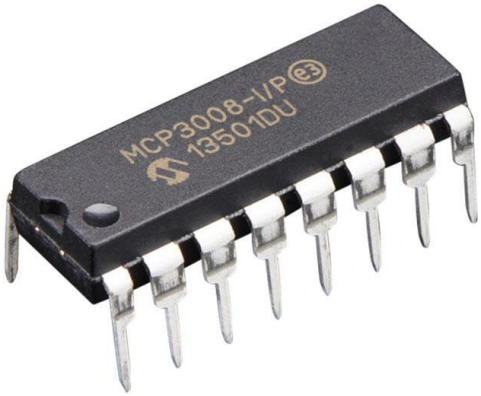

Figura 6. MCP3008-I/P

La conexión de este conversor con el sensor, la protoboard[5] y la Raspberry-Pi se verá en el Anexo C y sus especificaciones técnicas se encuentran en el siguiente Datasheet [22].

[5] Protoboard: placa de pruebas para interconectar componentes electrónicos.

2.1.7 Teléfono Android

Para probar las funcionalidades del proyecto se ha usado un Smartphone Android en el que se ha instalado la aplicación desarrollada, *ControlYourAnimals*. Concretamente se ha utilizado un Nexus 5, Figura 7, con las siguientes características principales [2]:

Figura 7.Nexus 5.

- Procesador de cuatro núcleos Qualcomm Snapdragon 800 a 2.26 GHz.
- 2 GB de memoria RAM.
- 16 o 32 GB de almacenamiento interno (sin posibilidad de expansión por MicroSD.
- Unidad gráfica (GPU) Adreno 330 de 450 MHz.

Debido a que la App es solo una versión de prueba con poco consumo de recursos podría instalarse en cualquier Smartphone con versión de Android posterior a la 4.0 o en un emulador para ordenador como por ejemplo *"BlueStacks"*.

2.2 Recursos Software

2.2.1 Máquina Virtual Ubuntu

Se ha utilizado el software Virtual-Box con una máquina virtual Ubuntu 16.04. En ésta se encuentran los Docker con el Context-Broker, la BBDD MySQL y Cygnus. También se ha alojado en esta Máquina Virtual el Servicio Web Spring que envía y accede a los datos. A continuación, se detallará algunas herramientas software instaladas en esta máquina.

2.2.2 MySQL-Workbench

MySQL-Workbench es una herramienta para visualizar bases de datos. Ésta se ha usado en el proyecto para visualizar la correcta creación de la base de datos y la inserción de la información. Esta herramienta también permite al desarrollador de la máquina diseñar, modelar, generar y administrar bases de datos [3].

2.2.3 Spring Tools Suite 4

Spring Tools Suite (STS) es un IDE basado en la versión Java EE de Eclipse, pero específicamente diseñado para trabajar con Spring Framework. Se ha utilizado la versión 4.

Algunas de las características destacadas son [4]:

- Soporte para Spring 3
- Asistente para la creación de proyectos Spring.
- Herramientas para la gestión de beans.
- Editores gráficos de archivos de configuración de Spring.
- Herramientas de desarrollo para Spring Web Flow y Spring Barch.

2.2.4 Sublime

Se ha utilizado Sublime como editor de texto para el código, debido a su sencillez y a la forma de adaptar el estilo de la fuente al lenguaje de programación que se está usando en ese momento. En el proyecto se ha usado Python, Java y CSharp con Sublime.

2.2.5 PostMan

Postman es un entorno de desarrollo de APIs que nos permite diseñar, probar y monitorizar servicios REST. Dispone de un entorno de desarrollo muy intuitivo, que te permite hacer peticiones muy cómodamente. Se profundizará en su funcionamiento en apartados posteriores.

Figura 8. Logo Postman (2018) [5].

En el proyecto que nos abarca ha sido realmente útil, tanto para crear las Entidades[6] en Fiware Context Broker, como para comprobar las peticiones a la API Rest del Servicio Web Spring. Se ha usado como herramienta de depuración a lo largo de la realización de todo el proyecto.

2.2.6 Unity

"Unity es un motor de videojuego multiplataforma creado por Unity Technologies. Unity está disponible como plataforma de desarrollo para Microsoft Windows, OS X, Linux. La plataforma de desarrollo tiene soporte de compilación con diferentes tipos de plataformas (Véase la sección Plataformas objetivo). A partir de su versión 5.4.0 ya no soporta el desarrollo de contenido para navegador a través de su plugin web, en su lugar se utiliza WebGL. Unity tiene dos versiones: Unity Professional (pro) y Unity Personal" [6].

[6] Entidad: productores/consumidores de contexto dentro o fuera de la plataforma Fiware.

Figura 9 .Logo Unity [8].

En el proyecto se ha utilizado "*Unity Personal"* para desarrollar la aplicación Android con la que el gestor controlará los animales. Se ha utilizado este entorno porque, a pesar de que está diseñado para el desarrollo de videojuegos, ofrece la posibilidad de desarrollar aplicaciones multiplataforma con multitud de funcionalidades más allá de los mencionados videojuegos. Con este motor se ha realizado de forma sencilla una aplicación animada (detallada en el Apartado 6) usando C# como lenguaje de programación [7].

3 TECNOLOGÍAS UTILIZADAS

Cualquier tecnología suficientemente avanzada es equivalente a la magia.

- Sir Arthur C. Clarke -

Este capítulo explica las tecnologías utilizadas en el desarrollo del proyecto. La instalación de estas herramientas se puede encontrar en los Anexos, mientras que el uso que se les ha dado vendrá explicado en los apartados *"Desarrollo de la Aplicación"*.

3.1 Docker

"Docker es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos" [9].

Figura 10. Logo de Docker [10].

En la máquina virtual usada para la realización del trabajo se ha creado diversos contenedores dockers para alojar el Context Broker, la base de datos de MongoDB, Cygnus y la base de datos MySQL. La instalación de esta tecnología se verá posteriormente en el Anexo A. Para facilitar la creación de tantos contenedores hemos usado una herramienta denominada Docker Compose, la cual utiliza un archivo YAML para configurar las distintas aplicaciones.

3.2 Fiware Orion Context Broker

"La plataforma FIWARE se conforma de componentes de plataforma de código abierto que se puede ensamblar junto con otros componentes de plataforma de terceros para acelerar el desarrollo de soluciones inteligentes." [11]

Orion Context Broker es una implementación en C++ de la API REST NGSIv2 desarrollada para la plataforma FIWARE para la administración de Contexto. El Context Broker es un componente obligatorio para cualquier solución de esta plataforma. La función principal de éste es la de administrar la información de contexto, lo que permite realizar actualizaciones y acceder al mismo [12]. A continuación, podemos ver una figura explicativa de las capas de abstracción en las que se divide Fiware, Figura 11.

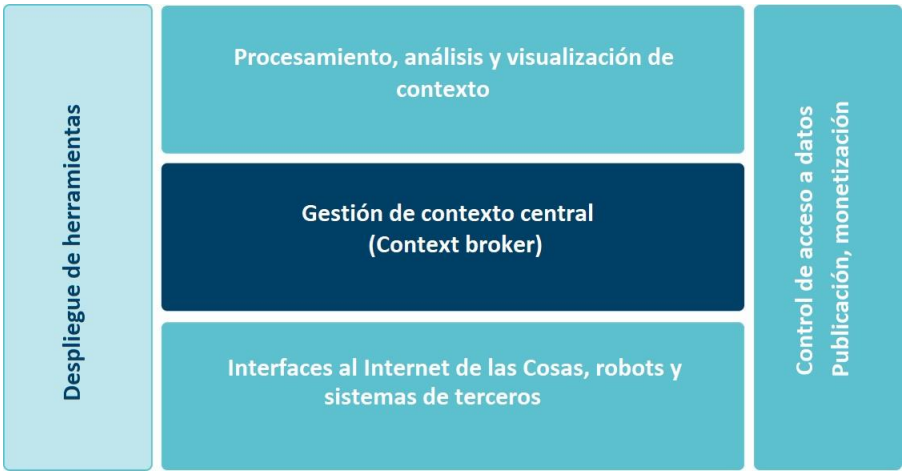

Figura 11. Estructura de la Plataforma Fiware [11].

- Se ha usado el estándar que propone Fiware para recopilar, gestionar y publicar información relacionada con la Acuariofilia y cría de reptiles. Usando el formato basado en entidades y atributos que propone, se obtiene y se almacena la información de forma que facilita su gestión y monitorización, Anexo A: Instalación y configuración de Context Broker con Docker.

3.3 Cygnus

Cygnus es un contenedor encargado de mantener ciertas fuentes de datos en almacenes de terceros que se hayan configurado. Permite tener un historial de los datos que se han recabado. Internamente Cygnus se basa en Apache Flume y ejecuta lo que se denomina un agente Flume específico por fuente de datos. Un agente Flume está compuesto básicamente por un oyente, un canal donde la fuente coloca los datos, y un sumidero que toma los eventos Flume del canal para almacenarlos en terceros. Permite almacenar los datos de contexto NGSI en diversas herramientas como, por ejemplo: MySQL, DynamoDB, Kafka, PostgreSQL, HDFS … [13]

Durante la tarea, se ha usado Cygnus para almacenar los datos de Fiware en una base de datos MySQL alojada en otro contenedor, en la misma máquina virtual. A esta máquina virtual accederá el Servicio Web desarrollado con Spring.

3.4 MySQL

MySQL es un Sistema de gestión de bases de datos relacional, colección de elementos de datos organizados en un conjunto de tablas con columnas y filas formalmente descritas. Es código abierto y está desarrollado por Oracle [14].

Figura 12. Logo de MySQL [14].

En esta base de datos se han almacenado los datos de cada entidad en una tabla, correspondiente cada una de éstas a una tienda de animales. En cada tabla se incluyen los datos recolectados por cada sensor en los distintos acuarios/terrarios. De esta forma tendremos en la BBDD un histórico de los valores que se han ido recabando.

3.5 Spring

"Spring es un framework para el desarrollo de aplicaciones y contenedor de inversión de control, de código abierto para la plataforma Java" [18]. Se ha usado Spring Boot que es una herramienta que viene integrada en la aplicación Spring Tool Suite 4, que se ha explicado anteriormente. Esta herramienta permite construir y ejecutar aplicaciones y servicios en Spring más rápidamente.

Figura 13. Logo de Spring [18].

Se ha usado esta tecnología para crear el Servicio Web que accede a la base de datos e interactúa con la aplicación Android. La instalación y configuración de STS se detallará en el Anexo D: Instalación y configuración de STS.

3.6 JPA e Hibernate

Java Persistence API (JPA), es un standard del lenguaje Java para el mapeo objeto-relacional (ORM) que junto con Hibernate forman una herramienta para el mapeo de atributos entre una base de datos relacional y el modelo de objetos de una aplicación.

Figura 14. Logo de Hibernate [41].

Estas herramientas han sido utilizadas en el proyecto para mapear los datos de la BBDD MySQL. De esta forma el usuario accede al Servicio Web Spring con la App y éste le devuelve los datos correspondientes de la base de datos.

3.7 Lenguajes de programación utilizados

3.7.1 Java

Java es un lenguaje de programación de propósito general, tipado, portable, orientado a objetos… que permite el desarrollo de todo tipo de aplicaciones. Éste es comercializado por Oracle [17].

Dentro del trabajo realizado se ha usado en el desarrollo del Servicio Web. En el Apartado 5 se profundiza en el desarrollo de estas aplicaciones.

3.7.2 Python

Python es un lenguaje de programación interpretado, dinámico y multiplataforma que soporta tanto orientación a objetos como programación imperativa. Es administrado por la Python Software Foundation [16].

La recolección de datos de los distintos hábitats llevada a cabo en la Raspberry se realiza por medio de un programa codificado en este lenguaje.

3.7.3 C Sharp

C Sharp o C# es un lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET. Su sintaxis básica deriva de C/C++ y utiliza un modelo de objetos similar al de Java, aunque incluye mejoras derivadas de otros lenguajes [15].

La plataforma de desarrollo de aplicaciones móviles Unity permite la codificación en este lenguaje, por lo que se ha utilizado para la programación de la App Control Yours Animals.

4 APLICACIÓN DESARROLLADA: DISPOSITIVO Y CONTEXT BROKER

The question of whether a computer can think is no more interesting than the question of whether a submarine can swim.

- Edsger W. Dijkstra -

En este capítulo vamos a concretar cómo se ha desarrollado la aplicación que ejecutará el Agente IoT, así como la configuración de los dispositivos conectados y la configuración del Context Broker.

4.1 Dispositivo

En los siguientes apartados se tratará como Dispositivo al conjunto de sensores y la Raspberry-Pi, así como el programa desarrollado en Python para la recolección de datos y actualización del Context Broker. No se entrará en detalles entre la comunicación sensores-Raspberry.

4.1.1 Raspberry Pi

Este dispositivo es el encargado de recoger los datos y enviarlos al Context Broker. La idea principal se basa en que se instale uno de éstos en cada tienda a monitorizar por el Gestor. Conectado a la Raspberry estarán los sensores necesarios para todos los hábitats de animales que haya en esa tienda.

Como se comentó anteriormente en este proyecto solo obtenemos los datos de dos sensores cuya conexión se mostrará con más detalles en el Anexo B: Configuración y conexión de los sensores con Raspberry Pi. En este dispositivo se estará ejecutando continuamente el programa que describiremos a continuación.

4.1.2 Sensores y MCP3009

Tenemos dos tipos de sensores de dos tecnologías distintas:

- El sensor DHT11, que es un sensor digital que va conectado directamente a los pines GPIO de las Raspberry y alimentado a 3.3V. Con una librería mostrada en el apartado siguiente se obtiene los datos de temperatura y humedad de forma sencilla.
- El sensor de PH, éste en concreto es un sensor analógico, que irá alimentado a 3.3V y conectado directamente al conector como se muestra en la Figura 52 del Anexo B. El conversor proporciona un voltaje como salida. Este voltaje se corresponde con un determinado nivel de PH. Algunos medidores

de PH vienen calibrados y te proporciona una tabla que relaciona la salida con el PH. En nuestro caso no se dispone de esta información por lo que hemos tenido que hacer un estudio empírico para obtener un nivel válido (entre 0-14). Para calibrar el instrumento y poder obtener una ecuación lineal que relacione Voltaje con el nivel de PH lo más aproximada posible se han usado los siguientes elementos con PH conocidos:

Líquidos	Voltaje (V)	Nivel de PH
Agua embotellada	1.225	7.6
Agua con dos cucharadas de bicarbonato	1.249	7.9
Leche	1.07	6.5
Vinagre	0.51	3.0
-	0.0	0

Tabla 1. Experimento Niveles de PH.

Con la información obtenida anteriormente se procedió a crear la recta de regresión que minimiza la distancia a esos puntos. Usando la fórmula de esa recta para programar el código que realiza la conversión en la Raspberry. A continuación, podemos observar un gráfico ilustrativo, Figura 15.

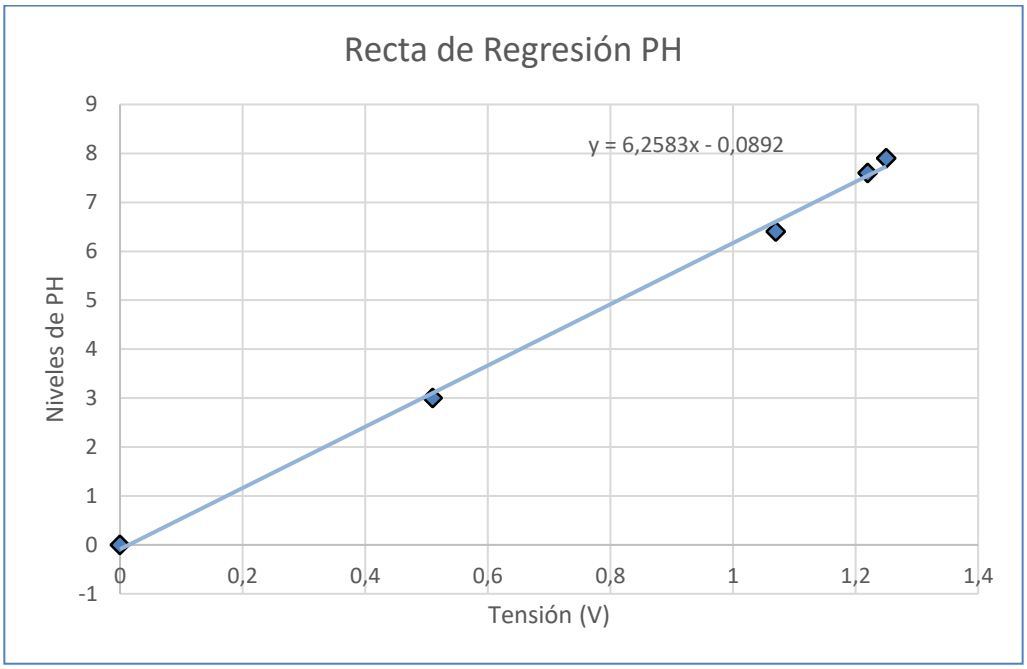

Figura 15. Recta de niveles de PH.

En el siguiente apartado se analizará cómo se ha implementado esto en el programa desarrollado para la RaspBerry.

4.1.3 Daemon_sensors.py

Este programa se ha desarrollado con el principal objetivo de que almacene la información proporcionada por los sensores y la envía al Context Broker de Fiware. También se ha creado en este programa una variable *"cerradura"* que simula como si hubiera conectada una cerradura Wifi. A través de la aplicación se puede actuar y abrir esta cerradura que como se comentó en la introducción es muy importante cuando se trata de animales peligrosos.

A continuación, se va a detallar los aspectos más interesantes del código y su funcionamiento.

- En primer lugar, se lleva a cabo la configuración de los puertos GPIOs a los que están conectados el sensor de temperatura/humedad y el conversor ADC. El esquema de conexión se encuentra en el Anexo C.
- Se ha creado un ciclo que va ejecutando el programa continuamente. En este bucle:
    - Se leen de los sensores la humedad y la temperatura, así como el dato de tensión que proporciona el sensor de PH. Para ello se han usado las librerías de Adafruit [23].

```
humedad, temperatura = Adafruit_DHT.read_retry(sensor, pin)
cerradura = "cerrada";
ph = 6.5126*chan.voltage-0.369
```

    - Una vez que se obtienen los datos procedemos a formar los Json para actualizar los atributos de la entidad correspondiente en el Context-Broker.

- También es necesario configurar la cabecera y la URL antes de realizar la petición PATCH para actualizar los datos. Como se explicó anteriormente solo actualizaremos los datos de un terrario y un acuario debido a que solo tenemos dos sensores instalados.

```
cabecera = {'Content-Type': 'application/json','Fiware-Service':'openiot','Fiware-ServicePath':'/'}
 #Se realiza la peticion para actualizar el terrario1
  req= requests.patch('http://192.168.1.110:1026/v2/entities/Tienda003/attrs',data =
json.dumps(data2),headers=cabecera);
  #Se realiza la petición para actualizar los datos del acuario1
  req= requests.patch('http://192.168.1.110:1026/v2/entities/Tienda003/attrs',data =
json.dumps(data3),headers=cabecera);
   print("Se ha actualizado los datos en Fiware");
```

- Por último, se realiza una consulta GET al context broker y se imprime el valor de la "cerradura" actualizada, esto se utiliza para comprobar que el actuador de la app Android ha funcionado correctamente.

```
cabecera2 = {'Fiware-Service':'openiot','Fiware-ServicePath':'/'}
req1=requests.get('http://192.168.1.110:1026/v2/entities/Tienda003
/attrs/terrario1/value',headers=cabecera2);
```

En el punto posterior se detallará por qué se han usado esos valores en la cabecera y en la URL, así como la estructura de las entidades y sus atributos.

4.2 Context Broker

Como se mencionó en apartados anteriores, el Context Broker de Fiware es un componente de esta plataforma para administrar información de contexto. Éste se ha instalado en un contenedor Docker como se explica en el Anexo A: Instalación y configuración de Context Broker, Cygnus y MySQL con Docker.

Este componente trabaja con entidades que están compuestas de atributos, que se encuentran en formato JSON y que se organizan en secciones dentro de la plataforma Fiware Orion en función de la URL.

Nuestro Context Broker estará accesible en la dirección localhost de la máquina virtual y en el puerto 1026. Para acceder a ésta desde el exterior de la máquina virtual hay que utilizar la dirección asignada a la interfaz de internet. La configuración de la máquina virtual viene detallada en Anexo A.

4.2.1 Entidades

Cada *entidad* creada se corresponde con una tienda de animales. En la visión general del proyecto un mismo Gestor podrá acceder a varias tiendas, teniendo éstas distintos *atributos* que se corresponden con los hábitats disponibles. Para nuestra realización práctica solo hemos creado la *entidad* que se muestra a continuación, Figura 16.

Figura 16. Entidad Context-Broker.

En la figura anterior tenemos la entidad que se ha usado para representar a la Tienda003. Esta entidad tiene los siguientes campos:

- id: es un identificador único para cada entidad. En el proyecto se ha decidido usar *"Tienda0XX"* para cada tienda.

- type: define un tipo para esa entidad, junto con el id permite identificar unívocamente a la entidad. Este atributo no es único y lo comparten todos los establecimientos con las mismas características.
- Atributos: los atributos están formado por dos campos. El campo *type*, que identifica el tipo y el campo *value*. En este último almacenaremos los datos a monitorizar. en nuestro caso vamos a tener los siguientes:
    - Acuario1: este atributo tendrá los datos que captan los sensores supuestamente colocados en un acuario y conectados a la Raspberry. Estos son la humedad, la temperatura y el nivel de PH en el agua. Es del tipo Acuario y en su campo value almacena estos datos.
    - terrario1: este atributo hace referencia a la existencia de un terrario que contiene un animal peligroso (*), teniendo éste un tipo distinto, *"Terrario_Peligroso"*. Se usa este tipo para discriminarlo del resto de terrarios no peligrosos. Estos hábitats suelen tener una cerradura que impide que sea abierto por personas no cualificadas. Para simular una cerradura se ha creado un campo cerradura en *value* que representa si está abierta o cerrada. De este modo el Gestor que tenga la App Android tiene la posibilidad de abrir la cerradura de un animal peligroso cuando lo considere preciso. Cambiará el valor de ese atributo y el programa ejecutado en la Raspberry actualiza el valor de la variable "*cerradura*". Este programa cerrará automáticamente la cerradura tras unos segundos de espera. Esta cerradura es solo ilustrativa, pero es fácilmente implementable con una cerradura wifi conectada a la Raspberry-Pi [24].
    - terrario2: este es del tipo "Terrario", simula un terrario con un animal normal, sin cerradura y almacena la temperatura y la humedad. En este caso son números enteros prefijados, no están relacionados con ningún sensor real.
    - terrario3: este comparte los mismos campos que el terrario 2.
    - address: en este atributo de tipo *"PostalAddress",* se usará para almacenar la calle, la ciudad y el código postal donde está ubicada la tienda.
    - trabajador: es de tipo trabajador y en el campo "*value"* se almacena le trabajador responsable de la tienda en ese momento.

En el transcurso del desarrollo del proyecto se ha usado Postman para comprobar que se crean y se actualizan correctamente las entidades. Una consulta de una entidad se hace mediante una petición GET como la mostrada en la figura. La URL donde se encuentran las entidades es la siguiente http://192.168.1.10:1026/v2/entities si se consulta desde un ordenador externo, dentro de la red privada 192.168.1.0/24 o con http://localhost:1026/v2/entities si se consulta desde la misma máquina. Orion Context Broker permite consultar, crear o modificar un atributo concreto realizando peticiones GET, PUT o PATCH. Usando la misma URL mencionada anteriormente añadiéndole */attrs/elnombredelatributo.* A continuación, se muestra un resumen de la API REST NGSI-9 de las peticiones usadas para la realización del trabajo, Tabla 2. Para una información mucho más detallada consultar la siguiente referencia [25].

HTTP REQUEST	DEFINICIÓN
GET	Realiza una consulta a la entidad o atributo especificado en la URL
POST	Esta acción crea entidades y/o atributos. En la siguiente referencia, [26] se detalla los distintos usos.
PUT	Este tipo se usa para reemplazar los atributos en las entidades existentes, es decir, se eliminan todos los atributos seleccionados y se agregan los incluidos en la solicitud.

DELETE	Borra la entidad o atributo que indica la URL
PATCH	Este tipo de acción se utiliza para modificar atributos ya existentes. Si se usara para crear nuevas entidades o atributos generaría un error.

Tabla 2. Resumen API REST NGSI-9.

4.2.2 Subscripción a Cygnus

En el proyecto solo se ha realizado una subscripción en el Context Broker. Cygnus utiliza el mecanismo de notificación de subscripción de Orion Context Broker. Cygnus necesita ser notificado cada vez que cambian los atributos de ciertas entidades, para eso necesita suscribirse. En el Anexo A se detalla la instalación de Cygnus con Docker Compose.

La subscripción que se ha realizado es simple, con el objetivo de que se notifique a Cygnus de todos los cambios de contexto que se produzcan. Esto sucede cada vez que se crea una entidad (Tienda) o cada vez que se actualice un atributo de esa tienda, es decir, los parámetros recolectados de los terrarios o acuarios. A continuación, se observa cómo se ha realizado la subscripción. Figura 17, consulta realizada con Postman:

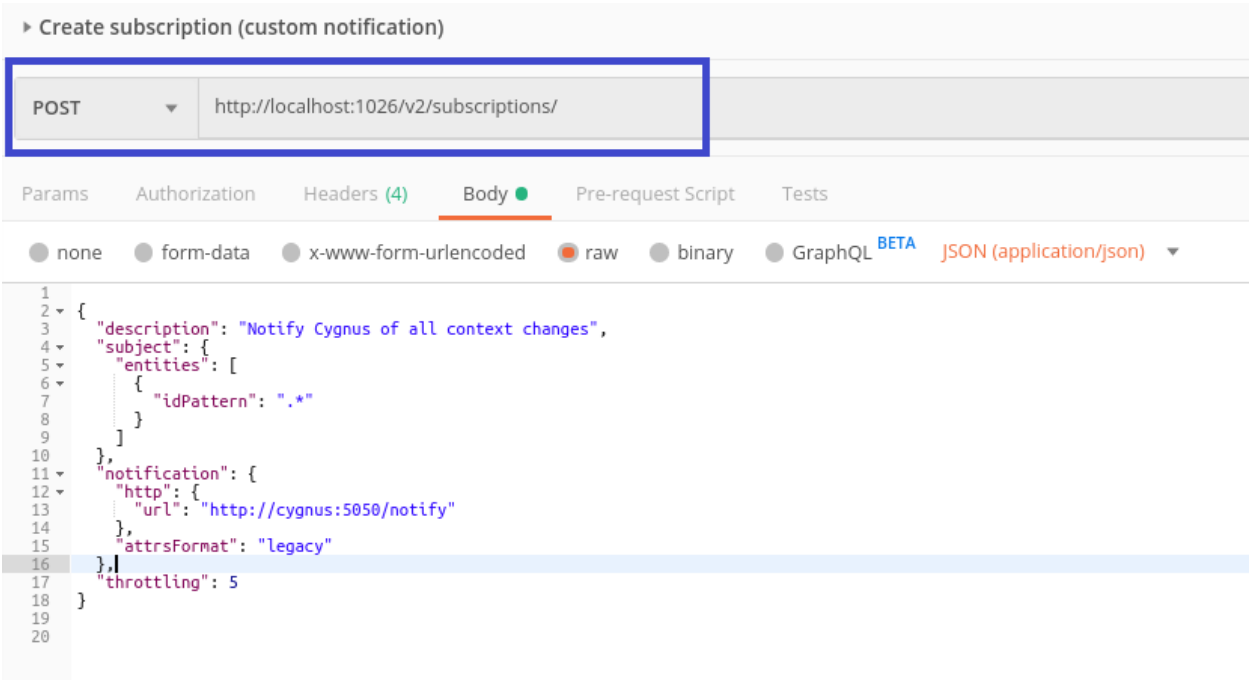

Figura 17. Context Broker-Subscripción Cygnus I

Se observa que para realizar una subscripción se utiliza una petición POST a la URL definida para esto, */subscriptions.* Analizando el cuerpo de la petición enviada podemos ver que es una subscripción para notificar a Cygnus, que se encuentra escuchando en el puerto TCP/5050. (El atributo idPattern se utiliza para indicar a qué entidades afectaría la suscripción. En este caso con *".*"* afecta a todas las entidades).

Cada subscripción tiene una serie de campos, algunos de ellos obligatorios y otros optativos. A continuación, se describen los utilizados en nuestro proyecto.

- description: Texto para describir la subscripción.
- subject: contiene los campos para determinar a qué entidades se le aplica la subscripción.
    - entities: define las entidades, para ello usa dos parámetros:
        - idPattern: identificador de las entidades
        - typePattern: indica el tipo de la entidad, en nuestro caso no es necesario porque indica a todas las entidades del contexto.
- notification: detalla la información relativa a las notificaciones
    - http: se utiliza para especificar la ruta donde se enviarán las notificaciones. En nuestro caso a la herramienta Cygnus.
    - attrFormat: se usa para definir el formato de la notificación. Se ha utilizado "legacy", que significa que utiliza el mismo formato que las entidades y además es compatible con NGSIv1.
- throttling: define el tiempo en segundos entre dos notificaciones consecutivas.

Podemos comprobar las subscripciones activas en nuestro Context Broker como muestra la Figura 18.

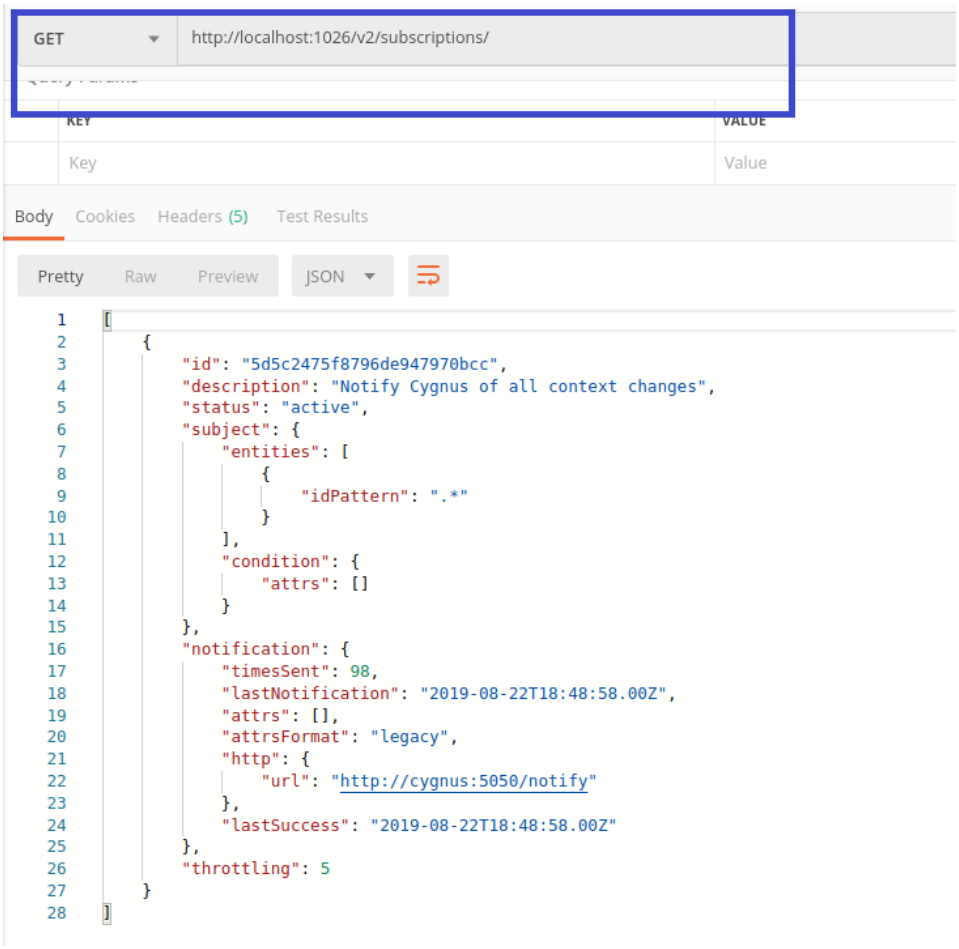

Figura 18.Context-Broker-Subscripción a Cygnus II

En la imagen anterior observamos distintos campos que aportan información a la subscripción como:

- TimeSent: número de subscripciones que ha enviado desde su creación.
- lastNotification: fecha y hora de la última notificación enviada.

- lastSuccess: fecha y hora de la última notificación exitosa.

Para eliminar la subscripción a Cygnus o cualquier otra basta con realizar una petición DELETE a la URL anterior indicando el *"id_subscripción"* de la subscripción a eliminar.

5 APLICACIÓN DESARROLLADA: SERVIDOR WEB Y BASE DE DATOS

Failure is an option here. If things are not failing, you are not innovating enough.

- Elon Musk -

En este apartado del trabajo se explica cómo se ha desarrollado el Servicio Web implementado, haciendo hincapié en el servidor de acceso a la base de datos y la API REST que usará la App Android para acceder a los datos. Para ello se detallarán las clases, métodos y atributos implementados, así como los ficheros de configuración necesarios. El código de este apartado se encuentra en el siguiente repositorio de GitHub [42].

5.1 Clases

Se procede a explicar las clases más importantes que se han desarrollado para la implementación del Servidor Web, estas se pueden observar en la Figura 19.

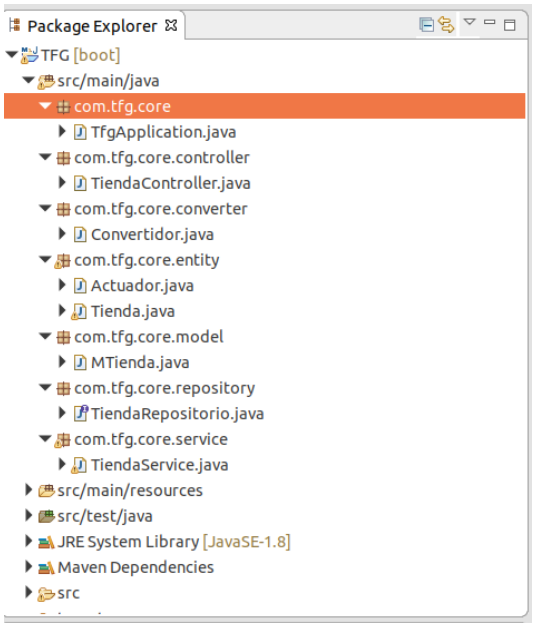

Figura 19. Estructura Servidor Web.

Antes de comenzar con la explicación de las distintas clases desarrolladas hay que tener en cuenta que el Context Broker crea la Base de Datos MySQL a la que este servicio accederá. Para realizar este acceso se ha utilizado la herramienta Hibernate como se detallará a continuación.

5.1.1 TfgApplication.java

Esta clase contiene el método main del programa y es el que ejecuta la clase *"TiendaController"* con la etiqueta *@RestController*.

La clase TfgApplication.java va etiquetada con la etiqueta @SpringBootApplication y es la primera en ejecutarse al iniciar el programa.

5.1.2 TiendaController.java

Esta clase contiene los métodos a los que se accederá con determinadas URLs. También tiene la etiqueta *@RestController,* que indica al sistema que es un controlador para una API REST y la etiqueta *@RequestMapping* que indica la versión que utilizaremos.

En nuestro proyecto las peticiones solo serán de consultas, por lo tanto, tendrán la etiqueta @GetMapping("url"), donde url es la dirección a la que el usuario debe acceder. En la Figura 20 se observan todos los métodos de esta clase.

```
@GetMapping("/tienda1")
public List<MTienda> obtener(){
    return service.obtener();
}
@GetMapping("/tienda/terrarios")
public List<MTienda> obtenerTerrarios(){
    return service.obtenerTerrarios();
}
@GetMapping("/tienda/numero/terrarios")
public int numero_terrarios(){
    return num_terrarios;
}
@GetMapping("/tienda/numero/acuarios")
public int numero_acuarios(){
    return num_acuarios;
}
@GetMapping("/tienda/acuarios")
public List<MTienda> obtenerAcuarios(){
    return service.obtenerAcuarios();
}
@GetMapping("/tienda")
public List<MTienda> obtenerTiendas(Pageable pageable){
    return service.obtenerPorPaginacion(pageable);
}
@GetMapping("/tienda/maximo")
public int maximo(){
    return service.Contar();
}
```

Figura 20. Métodos clase TiendaController.

Los métodos anteriores son llamados al realizar una petición Get a la URL indicada y éstos devuelven la salida del método correspondiente de la clase *TiendaServicio*, que se verá en el apartado siguiente. Hay que destacar el método obtenerTiendas el cual recibe como parámetro un objeto de tipo *"pageable"* para que devuelva un número de filas de la BBDD específico.

5.1.3 Servicio.java

Esta clase con etiqueta *@Service("servicio")* se encarga de obtener los datos que se muestran al realizar la consulta. La etiqueta anterior es una especialización de la anotación de componentes, que establece las clases de capa de servicios. Utiliza la interfaz creada *Repositorio* y posee los siguientes métodos principales:

- **public int Contar():** devuelve el índice de la última entrada de la tabla de la BBDD. Esto es útil para realizar la consulta desde la aplicación móvil. Se utiliza para obtener el dato más reciente, pero hay que tener en cuenta que Cygnus accede de forma concurrente a la Base de Datos, pudiéndose dar el caso de que la información devuelta no sea la más actualizada. Esto no afecta a la funcionalidad de nuestro proyecto porque se realizan consultas consecutivamente. Se explicará en profundidad en el apartado 6.3.
- **public List <MTienda> obtenerTerrarios ():** devuelve una lista que contiene los terrarios existentes de esa tienda.
- **public List <MTienda> obtenerAcuarios ():** devuelve una lista que contiene los acuarios existentes de esa tienda.
- **public List <MTienda> obtenerPorPaginacion (Pageable pageable):** devuelve una lista de un número de tiendas que depende del índice de paginación que se introduzca en la URL. Se verá un ejemplo de esto en el Apartado 5.3 Prueba de funcionamiento.

5.1.4 Tienda.java

Esta clase contendrá una entidad que tiene como atributos los campos que queremos almacenar. Se añaden las etiquetas @Table (name = "Tienda003_Tienda") y @Entity. Con estas etiquetas JPA se puede mapear los datos de la BBDD. Es importante que el nombre sea el mismo que el de la tabla MySQL ya creada por el

Context Broker para acceder a los datos de ésta.

Esta clase tendrá los siguientes atributos:

- Private String recvTimeTs: se utiliza para almacenar el instante de tiempo de su última actualización.
- Private String recvTime: empleado para almacenar el instante de tiempo desde su creación.
- Private String fiwareServicePath: almacena la ruta del Context Broker a la que accede.
- Private String entityId: almacena el identificador de la entidad.
- Private String entityType: el tipo de la entidad.
- Private String attrName: hace referencia al nombre del atributo.
- Private String attrValue: almacena el valor del atributo.
- Private String attrMd: es creado para que coincida con la tabla creada por el Context Broker, pero su valor carece de utilidad en nuestro proyecto.

Los atributos anteriores se corresponden con las columnas de la tabla asociada a la Tienda y deben contener la etiqueta *@Colum (name =" nombre _de _la _columna").* La tabla es creada y actualizada por la herramienta Cygnus comentada anteriormente. Podemos observar esta tabla con la ayuda de Mysql-Workbench en la Figura 21.

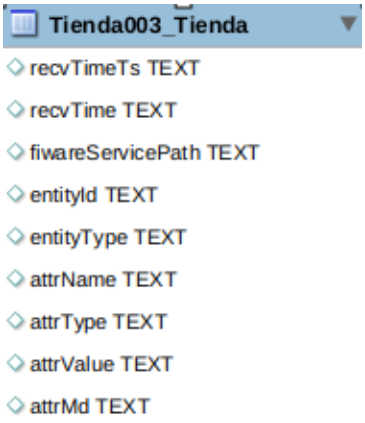

Figura 21. Entidad Tienda003.

Además de los atributos, esta clase también contiene los métodos Getters, Setters y un constructor para rellenar estos campos. Por ejemplo, en la Figura 22 podemos observar el diagrama de clases de ésta.

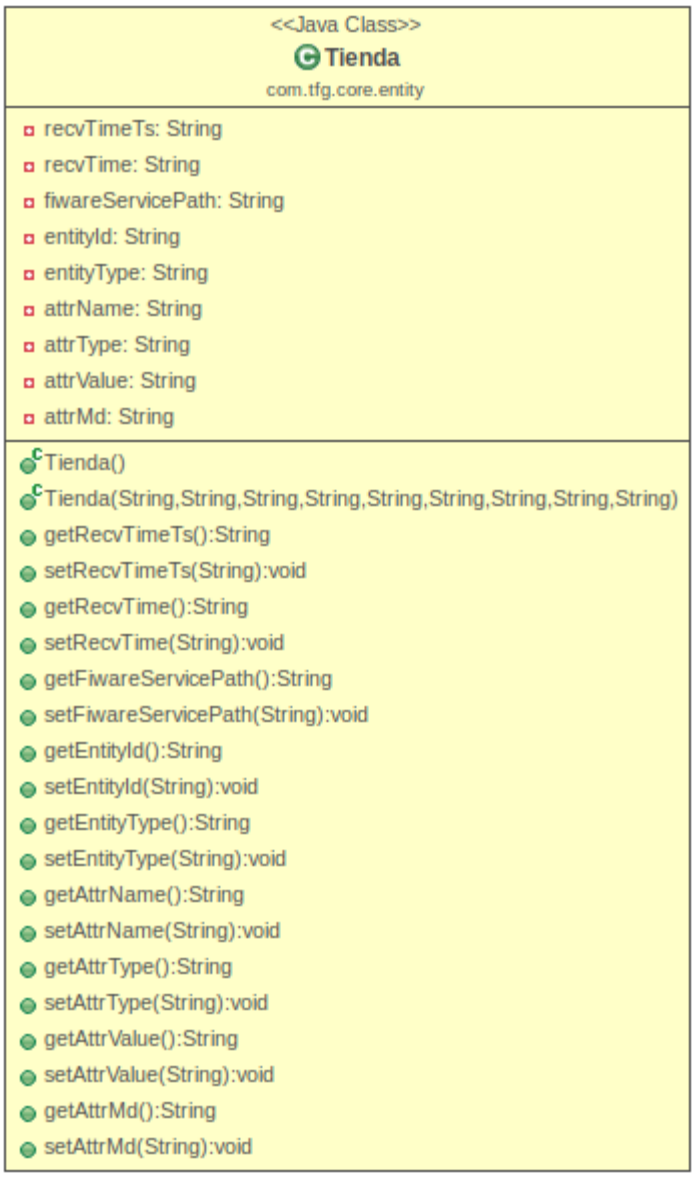

Figura 22. Diagrama de Clases I.

Hay que destacar que también se ha usado un modelo "*MiTienda.java"*, para evitar errores al acceder a los datos de la BBDD relacional. De esta forma un controlador nunca accederá a una entidad sino a su modelo asociado. Éste tendrá los mismos atributos que la Entidad. La única diferencia es que no tiene las etiquetas.

5.1.5 Convertidor.java

Esta clase se ha creado para hacer la conversión de modelo a entidad, de forma que permita trabajar en los repositorios con la entidad y en los controladores con el modelo. De esta forma se soluciona un error que aparecía al realizar consultas a la base de datos MySQL.

Solo dispone de un método "*convertirLista*" que recibe una lista de entidades ("*Tienda*") y devuelve su equivalente de modelo ("*MTienda*").

5.1.6 Repositorio.java

Repositorio.java es una interfaz que contiene la definición de los métodos que usa la clase Servicio para acceder a los datos almacenados en la Base de Datos. En la Figura 23 se muestra un diagrama de clases con las relaciones correspondientes entre ellas.

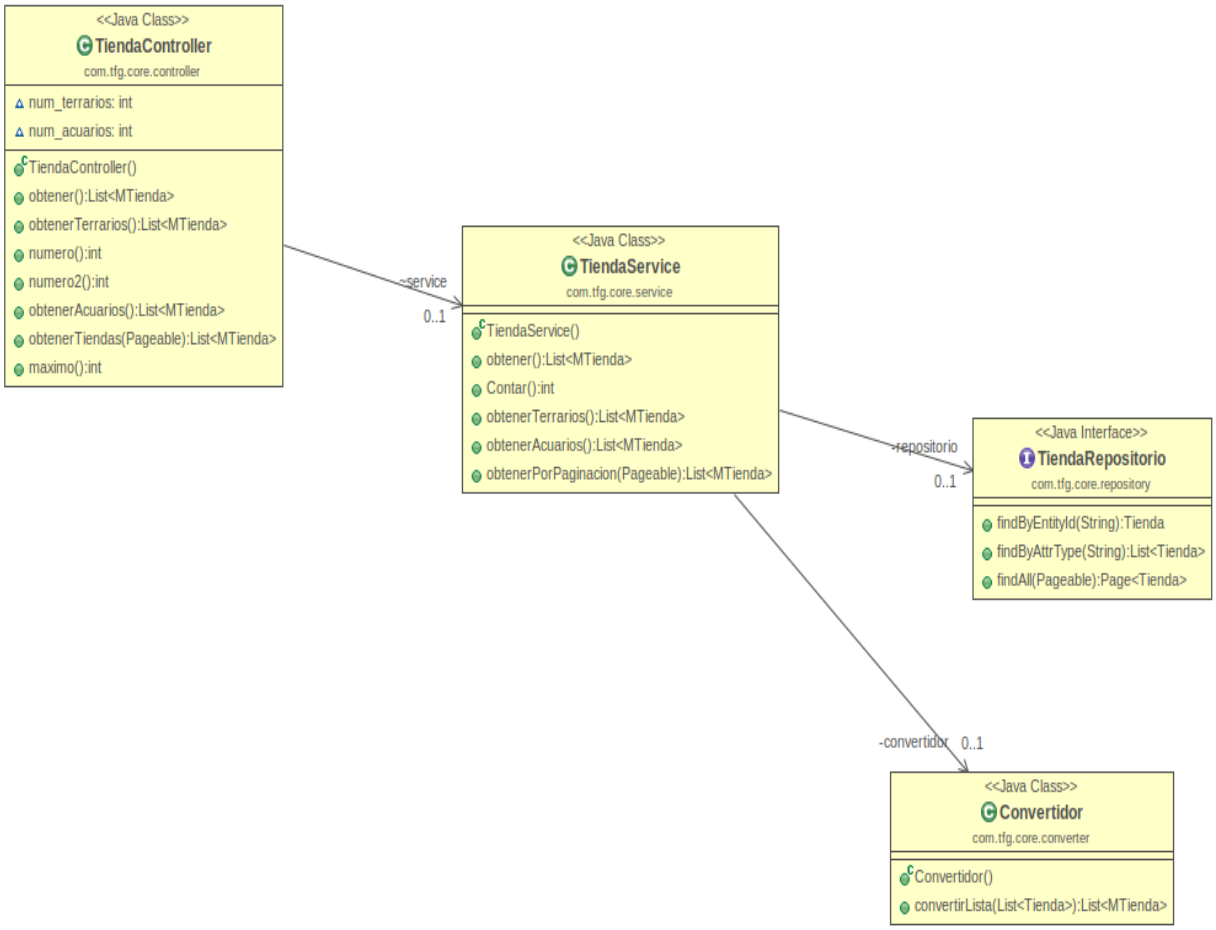

Figura 23. Diagrama de Clases II.

Los métodos de esta clase devuelven una o varias entradas de la tabla almacenada en la BBDD en función de un valor de un atributo. Por ejemplo, EntityId, AttrType, ALL (todos) …

5.2 Fichero de configuración

5.2.1 pom.xml

Este archivo contiene información sobre el proyecto y los detalles de configuración utilizados por Maven. Algunas de estas características se encuentran en la Figura 24.

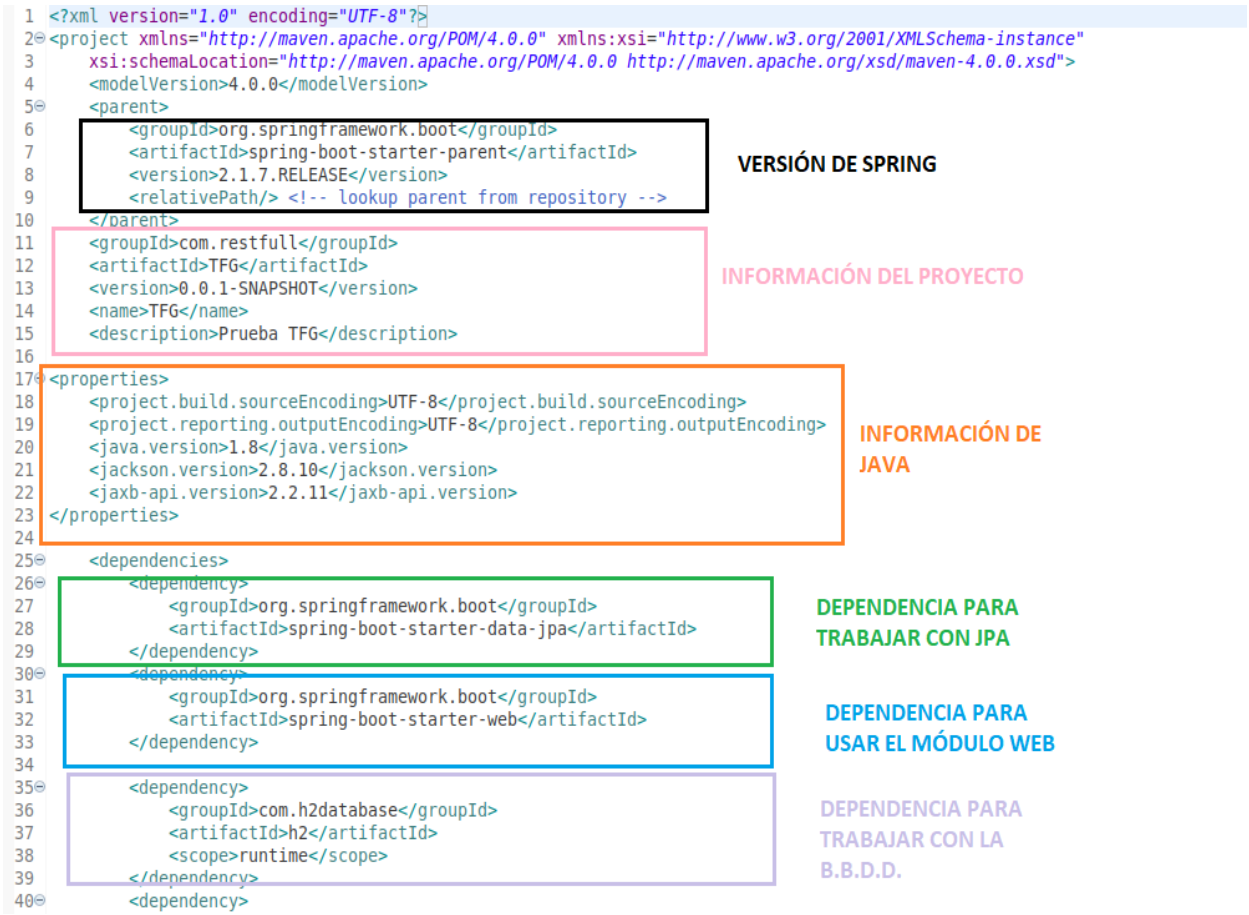

Figura 24. POM Spring.

5.2.2 application.properties

En este fichero se definen las propiedades de lo relativo a la base de datos MySQL y de Hibernate. A continuación, se citan estas características y se comentará su utilidad.

- En este fichero se incluye el número de puerto donde escucha el Servidor Web, la dirección donde se encuentra la BBDD. En nuestro caso está en el docker levantado por el Context Broker en la misma máquina.

  También hay que indicar el usuario y contraseña para poder acceder a la Base de Datos. Podemos ver estos campos en la Figura 25.

```
server.port=8090
spring.datasource.url = jdbc:mysql://localhost:3306/openiot?useSSL=false
spring.datasource.username = root
spring.datasource.password = usuario
```

Figura 25. application.properties I.

- También se ha de incluir en este fichero información para la *"paginación"*, Figura 26. Es decir, para poder obtener un número determinado de filas de una tabla de la Base de Datos. En este fichero hemos definido el valor por defecto del número de páginas (rest.default-page-size) y el máximo (rest.max-page-size) que devolverá una consulta. También establecemos el nombre que se usará en la url (rest.page-param-name) y el nombre para establecer la limitación de éstas (rest.limit.param.name). Mirar la Figura 30 para ver cómo se reflejan las propiedades *"name"* en una consulta HTTP.

```
7
8  spring.data.rest.page-param-name=page
9  spring.data.rest.limit-param-name=limit
10 spring.data.rest.default-page-size = 5
11 spring.data.rest.max-page-size = 20
12
```

Figura 26. application.properties II.

- Por último, se ha añadido información para Hibernate, como el dialecto de la comunicación con la BBDD, estrategia de nomenclatura, actualización de la BBDD, etc.

5.3 API de Consulta de Datos

En este apartado se observan las distintas consultas (método GET) disponibles para el Servidor Web, las cuales se proceden a detallar en la Tabla 3.

URI	Parámetros	Descripción
http://192.168.1.110:8090/tienda	"page": número de página "size": tamaño	Consulta los datos de determinados hábitats de una tienda.
http://192.168.1.110:8090/tienda/acuarios	-	Consulta de los datos de los acuarios de la tienda.
http://192.168.1.110:8090/tienda/terrarios	-	Consulta de los datos de los terrarios de la tienda.
http://192.168.1.110:8090/tienda/numero/acuarios	-	Consulta del número de acuarios disponibles.
http://192.168.1.110:8090/tienda/numero/terrarios	-	Consulta del número de terrarios disponibles.

http://192.168.1.110:8090/tienda/maximo	-	Consulta del ultimo índice de la tienda.
http://192.168.1.110:8090/tienda1	-	Consulta de todos los datos almacenados de la tienda.

Tabla 3. API de consultas.

5.4 Prueba de funcionamiento

En este subapartado se realiza algunas pruebas de consulta con el Servidor Web. Éste debe de estar arrancado y se utilizan las herramientas Postman y MySQL-Workbench.

1. En esta prueba se va a comprobar que el número de terrarios es el correcto. En nuestro proyecto habrá tres terrarios y un acuario a modo de ejemplo. Para comprobar esto realizamos la consulta que se muestra en la Figura 27.

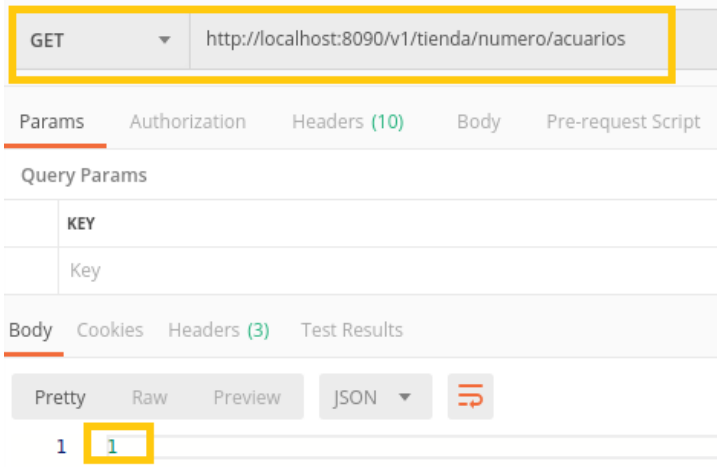

Figura 27. Consulta Servidor Web "Numero de Acuarios".

2. En la Figura 28 se muestra la consulta de las entradas almacenadas en la tabla de la tienda.

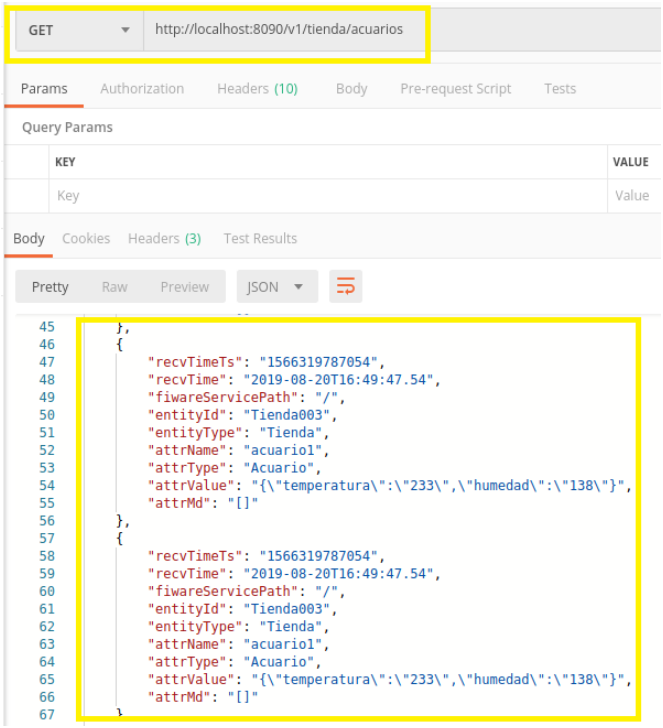

Figura 28. Consulta Servidor Web "Entradas Almacenadas".

3. Por último, mostraremos el uso de la paginación para obtener solo los datos de un determinado hábitat. Por ejemplo, podemos observar en la Figura 29, tomada de MySQL-Workbench, que la entrada número 527 se corresponde con la última actualización de los datos del terrario3.

#	recvTimeTs	recvTime	fiwareServicePath	entityId	entityType	attrName	attrType	attrValue	attrMd
526	1567096671651	2019-08-29T16:37:51.651	/	Tienda003	Tienda	terrario2	Terrario	{"temperatura":"233","humedad":"1...	[]
527	1567096671651	2019-08-29T16:37:51.651	/	Tienda003	Tienda	terrario3	Terrario	{"temperatura":"23","humedad":"13"}	[]

Figura 29. Entrada MySQL-Workbench.

Si se realiza ahora la consulta, indicando en la URL como índice de paginación la fila de la tabla y como tamaño que solo queremos obtener una entrada, se observa que el resultado es el mismo que el mostrado anteriormente.

Podemos verificar lo anteriormente comentado con la ayuda de la Figura 30.

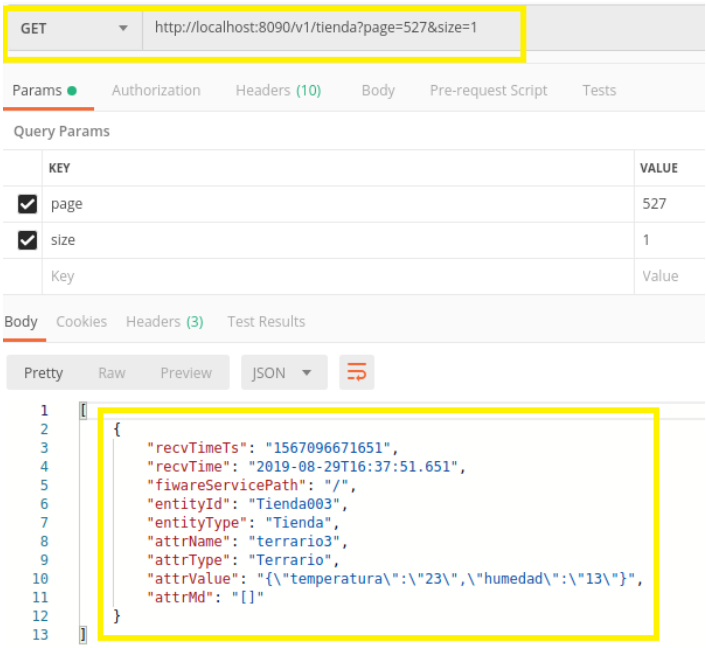

Figura 30. Consulta Servidor Web “Paginación”.

En las consultas anteriores observamos que va dirigida al puerto 8090, que se definió como el puerto habilitado para el servidor. Además, las URLs coinciden con la etiqueta @GetMapping como se explicó anteriormente. Las consultas se han enviado a la dirección local, pero se puede acceder también utilizando la dirección IP privada de la interfaz de la Máquina Virtual (Anexo E: Configuración de la Máquina Virtual).

6 APLICACIÓN DESARROLLADA: APLICACIÓN EN ANDROID E INTERFAZ DE USUARIO

Nothing is particulary hard if you divide it into small jobs.

- Henry Ford -

LA aplicación con la que el usuario final monitoriza y controla los hábitats ha sido desarrollada usando C# y la plataforma Unity. En los siguientes apartados se detallará este desarrollo. Dividiéndose en Escenas[7]. En cada una de éstas se explica los componentes de la interfaz gráfica que aparecerán y las características más importantes de los scripts utilizados.

Unity trabaja mediante la asignación de Scripts en C# con objetos UI de la interfaz gráfica. De esta forma conseguimos que los componentes de la interfaz sean controlados mediante el código. Además, a cada escena le hemos asociado un Script denominado *"Controlador"* que será el motor principal.

A continuación, en la Figura 31 se muestra el sistema de directorios creado para nuestro proyecto Unity.

Figura 31. Directorios del proyecto en Unity.

En estos directorios se almacenan los recursos usados como escenas, imágenes, prefabs[8] y scripts. Nuestra aplicación "Control Your Animals" está compuesta por tres escenas que se detallan en profundidad en los siguientes apartados.

A continuación, se muestra el icono de nuestra Aplicación en nuestro teléfono Android (Figura 32).

[7] Escenas: conjunto de entornos y componentes de una parte de la Aplicación.
[8] Prefabs: son objetos reutilizables desde los recursos del proyecto.

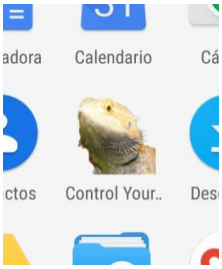

Figura 32. Logo App.

6.1 Escena 1: Tiendas disponibles

La primera escena que se ejecuta al abrir nuestra aplicación es la interfaz de usuario mostrada en la Figura 33.

Figura 33.Captura de pantalla Escena I.

En esta Escena se muestran las tiendas disponibles. En nuestro caso solo hemos implementado una tienda, pero si el gestor fuera el encargado de varias de ellas, aparecerían todas.

Esta escena dispone de un Canvas[9] muy sencillo, con un simple título al iniciar. Cuando se arranca la aplicación, comienza la ejecución del Controlador asociado a esta escena *Controller.cs*, el cual creará tantos Prefabs de botones como Tiendas disponibles. A modo de ejemplo implementaremos dos tiendas, pero ambas contendrán los mismos datos obtenidos por los sensores.

[9] Canvas: en el entorno Unity es el componente que hace referencia al área donde todos los elementos UI deben estar.

En la Figura 34 podemos observar cómo se crean los dos botones (Instancias de Prefabs).

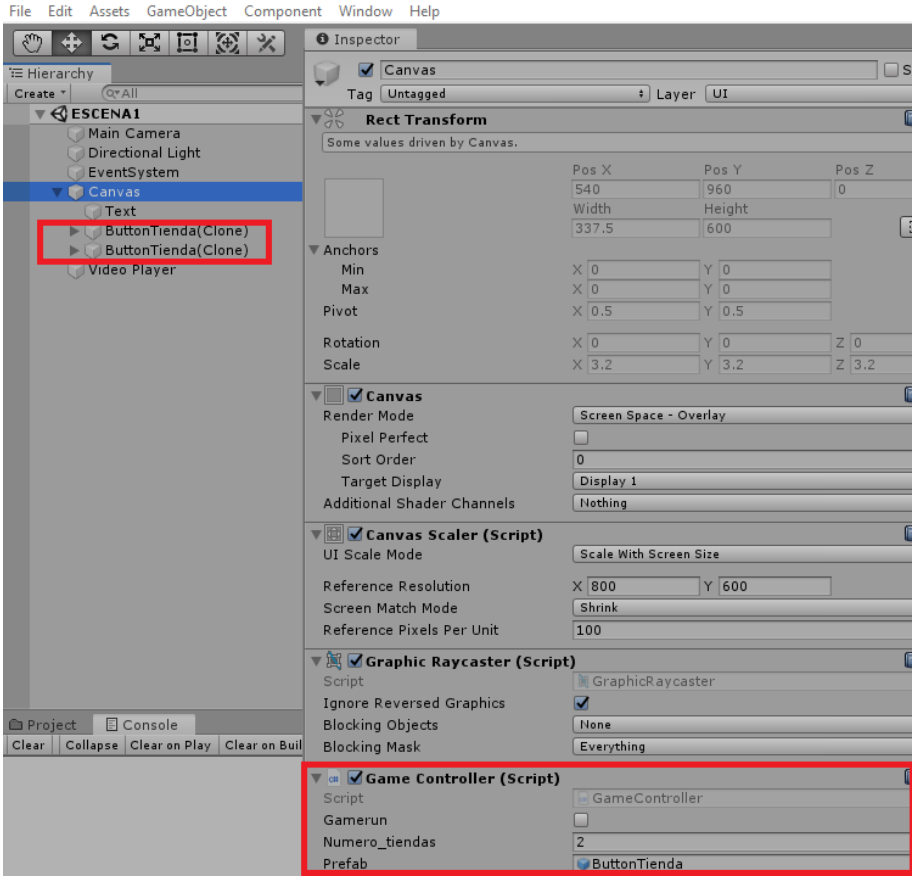

Figura 34. Unity I.

Se puede ver cómo el componente Canvas de esta Escena tiene asociado el Script GameController.cs. En este código se instancian estos botones cada vez que se inicia la App. Para entender esto basta con observar el siguiente fragmento de código de la clase GameController, Figura 35.

```
//Método que se ejecuta al arrrancar la escena (similar al main)
void Start () {

    numero_tiendas = 2;
    GameObject[] botones =new GameObject [numero_tiendas+1];//Creamos un array de objetos donde instanciaremos los prefabs del boton
    GameObject[] childText=new GameObject [numero_tiendas+1];//Creamos un array de objetos para el texto
    for (int i =1;i<=numero_tiendas;i++){
        botones[i] = Instantiate(prefab) as GameObject; //Vamos instanciando los Prefabs
        botones[i].transform.SetParent(this.transform);
        botones[i].transform.position= new Vector3 (550f,1700.0f-300f*i,0f); //Ajustamos la posición en la pantalla
        botones[i].transform.localScale=new Vector3(1.377f,0.98f,0f);
        childText[i]=botones[i].transform.GetChild(0).gameObject;//Hace referencia al componente Texto del Prefabs
        childText[i].GetComponent<UnityEngine.UI.Text>().text = "Tienda " + i;//Imprimimos
        }

}
```

Figura 35. Código GameController.cs

En el bucle de la figura anterior se muestra cómo se instancian los botones, se ajustan a la posición deseada en la pantalla y se actualiza el componente Text de estos.

Por otro lado, cada Prefabs "*Button"* solo tendrá asociado un Script con el método mostrado en la Figura 36 que se ejecutará cada vez que pulsemos el botón y ejecutará la escena correspondiente.

```
public class button : MonoBehaviour
{
    //Método que se ejecuta al pulsar sobre el botón
        public void button_on(){
            SceneManager.LoadScene ("ESCENA2");
        }

    }
```

Figura 36. Código ButtonTienda.cs

6.2 Escena 2: Hábitats disponibles

El aspecto de la interfaz de la segunda Escena es el mostrado en la Figura 37.

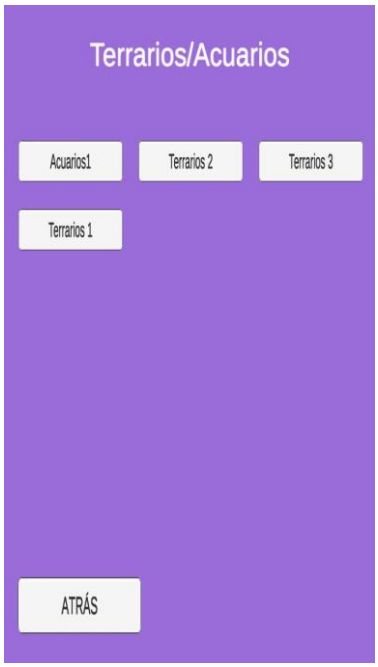

Figura 37. Captura de pantalla Escena II.

Ésta tendrá tantos botones como hábitats existan en nuestra tienda. Como se explicó anteriormente en nuestro ejemplo habrá cuatro hábitats, dos de ellos con datos ficticios y por otra parte el Terrario 1 y el Acuario 1 con los datos recolectados en tiempo "real" por los sensores.

La instancia de los prefabs de los botones es idéntica a la explicada en el apartado anterior. Teniendo en cuenta que en este caso la aplicación realiza una consulta HTTP para saber el número de terrarios y acuarios disponibles, esto se realiza gracias a la ayuda de la librería *"UnityWebRequests"*.

La clase Controller2.cs tendrá los siguientes métodos:

- Start(): El método start se ejecuta al inicio de la Escena y es el encargado de llamar al método GetText, pasándole como parámetros las siguientes URLs:
    - http://192.168.1.110:8090/v1/tienda/numero/acuarios
    - http://192.168.1.110:8090/v1/tienda/numero/terrarios

Ambas URLs con la dirección IP y puerto del Servidor Web.

- IEnumerator GetText (string url): recibe como parámetro la URL sobre la que se va a realizar la consulta. En este caso consulta el número de terrarios y/o acuarios. En la Figura 38 se muestra cómo se ha llevado a cabo a nivel de código.

```
        IEnumerator GetText(string url) {

        UnityWebRequest www = UnityWebRequest.Get(url);
        yield return www.SendWebRequest();

        if(www.isNetworkError || www.isHttpError) {
            Debug.Log(www.error);
        }
        else {
            // Muestra los datos como texto
            Debug.Log(www.downloadHandler.text);
            numero=int.Parse(www.downloadHandler.text);

            crearPeceras(numero);

            byte[] results = www.downloadHandler.data;

        }
    }
}
```

Figura 38. Consulta Get en Unity.

- Public void crearPeceras (int numero): Este método será llamado en la ejecución del método GetText y será el encargado de instanciar los botones de los acuarios y terrarios disponibles en la tienda. Creará tantos botones como indique el parámetro *"numero"* y se desarrolla de igual forma que en la Escena 1.

Además, se ha creado un botón *"Atrás"* en la interfaz. Este nos permite volver a la Escena anterior cada vez que es pulsado. El script asociado a este elemento *"Botón"* es similar que el de la Figura 36.

6.3 Escena 3: Información

Una vez que se selecciona el hábitat que queremos monitorizar o controlar se carga la Escena 3 de nuestra aplicación en la que se encontrará la información relativa a los distintos hábitats. En nuestro proyecto tenemos dos tipos de terrarios y uno de acuario. A continuación, podemos observar la información que se muestra en cada uno de éstos.

- Terrario: el tipo terrario hace referencia a los terrarios con animales no peligrosos, que no precisan de cerradura. En este caso solo aparece en pantalla los valores de humedad y temperatura. En la siguiente foto (Figura 39) observamos la información correspondiente al Terrario 3, perteneciente a este tipo.

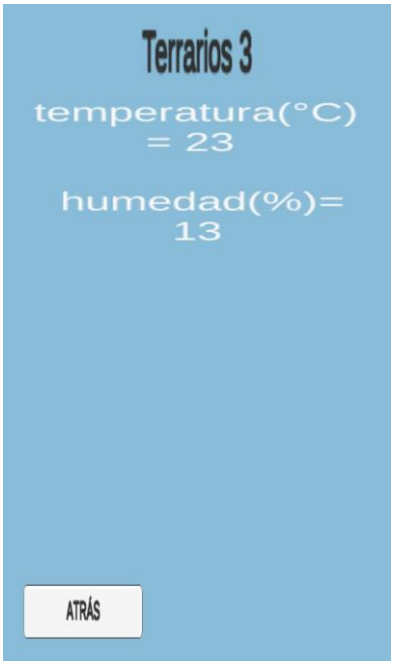

Figura 39. Captura de pantalla del Terrario 3.

- Terrario Peligroso: en los terrarios que contienen animales venenosos o agresivos aparece un nuevo campo relacionado con el estado de la cerradura. Además, en este caso se incluye un botón "Abrir" que al ser pulsado actúa sobre la Raspberry Pi poniendo el valor de la variable cerradura en "abierto". Podemos observar este cambio con respecto al tipo anterior en la Figura 40.

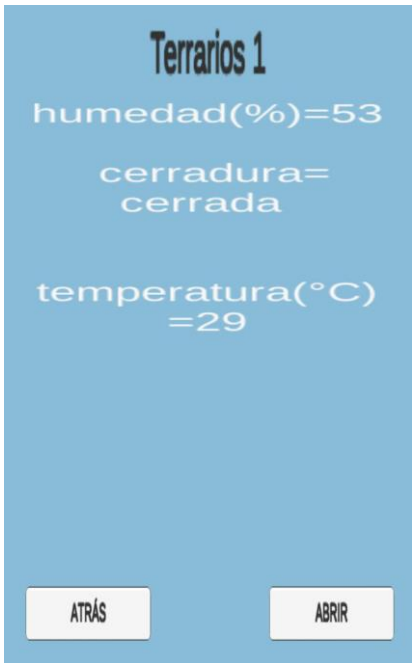

Figura 40. Captura de pantalla Terrario 1.

- Acuario: este tipo hace referencia a animales acuáticos y muestra la temperatura y el nivel de ph del agua. A continuación, se muestra un ejemplo (Figura 41).

Figura 41. Captura de pantalla del Acuario 1.

Esta Escena posee como script principal *Controller3.py*, el cual funciona de manera muy similar al mostrado en el apartado anterior. Éste dispone de los siguientes métodos:

- Start(): método que se ejecuta al inicio del programa y en su interior llama al método GetMaximo.
- GetMaximo(): método que realiza una petición GET a la dirección http://192.168.1.110:8090/v1/tienda/maximo. Esta petición devuelve el último valor introducido en la BBDD y después llama a GetText ().
- GetText(): este método es el encargado de obtener la información de la tabla e imprimirla por la pantalla de nuestro móvil. Para obtener dicha información realiza la petición GET a http://192.168.1.110:8090/v1/tienda?page="+rows_tabla+"&size=1. "*Rows_tabla"* es un índice que identifica a cada uno de los hábitats disponibles.

  Recibe los datos en un formato JSON y se procesa con la ayuda de la librería JsonUtility. Este procesamiento se hace de la siguiente manera:

  - Se crea un objeto con tantos atributos como tenga el JSON recibido. En nuestro caso hemos desarrollado el siguiente objeto, Figura 42.

```
[System.Serializable]
public class Terrario
{

    public string recvTimeTs;
    public string recvTime;
    public string fiwareServicePath;
    public string entityId;
    public string entityType;
    public string attrName;
    public string attrType;
    public string  attrValue;
    public string attrMd;
```

Figura 42. Clase Terrario.

Cada atributo de esta clase se corresponde con las columnas de la tabla de la Base de Datos explicada anteriormente en el apartado relativo a STS.

- Realizamos la conversión del Json al objeto mencionado. Para ello usamos el siguiente método, Figura 43.

```
ter = JsonUtility.FromJson<Terrario>(NewString);
```

Figura 43. Conversor JSON.

La sentencia anterior transforma el String que contiene el JSON devuelto por la petición GET en un objeto Terrario llamado ter. Después de esta asignación podemos acceder a los atributos de este método y representarlos de manera sencilla.

- Update(): es uno de los métodos de Funciones de Eventos que ofrece Unity [40] y se ejecuta continuamente. En nuestro caso, cada vez que expire un Timer de 5s llama a GetMaximo. De esta manera actualiza la información obtenida.

Hay que tener en cuenta que el Servidor Web y Cygnus consultan los datos de forma concurrente pudiendo ocurrir que cuando la aplicación Android reciba los datos, éstos ya se hubieran actualizado en la BBDD. Esto no afecta a la funcionalidad del Proyecto debido a dos factores:

- Los parámetros medidos por los sensores no sufren cambios bruscos significativos.
- Cuando expire el siguiente timer en 5s volvería a consultar la tabla y actualizará los datos en la APP.

Algunas de estas consultas se muestran en la Figura 44, obtenida con la herramienta Wireshark.

```
863 66.784428   192.168.1.106   192.168.1.110   HTTP   276 GET /v1/tienda/maximo HTTP/1.1
865 66.787961   192.168.1.110   192.168.1.106   HTTP    72 HTTP/1.1 200   (application/json)
868 66.823552   192.168.1.106   192.168.1.110   HTTP   286 GET /v1/tienda?page=1086&size=1 HTTP/1.1
870 66.828451   192.168.1.110   192.168.1.106   HTTP    72 HTTP/1.1 200   (application/json)
920 68.281033   192.168.1.106   192.168.1.110   HTTP   285 GET /v1/tienda/numero/acuarios HTTP/1.1
922 68.282537   192.168.1.110   192.168.1.106   HTTP    72 HTTP/1.1 200   (application/json)
```

Figura 44.Captura Wireshark GET.

El botón Atrás que aparece en la Interfaz funciona exactamente igual que el explicado en el apartado anterior. Sin embargo, en el caso de un terrario de tipo Terrario Peligroso el botón Abrir ejecuta el siguiente código cada vez que es pulsado (Figura 45):

```
public void button_on(){


        StartCoroutine(PutText());
    }
    IEnumerator PutText() {


    UnityWebRequest www = UnityWebRequest.Put("http://192.168.1.110:1026/v2/entities/Tienda003/attrs/terrario1","{ \

    www.SetRequestHeader("Fiware-Service", "openiot");
    www.SetRequestHeader("Fiware-ServicePath", "/");
    www.SetRequestHeader("Content-Type", "application/json");
    yield return www.SendWebRequest();

    if(www.isNetworkError || www.isHttpError) {
        Debug.Log(www.error);
    }
    else {
        // Show results as text
        Debug.Log(www.downloadHandler.text);

        // Or retrieve results as binary data
       // byte[] results = www.downloadHandler.data;
    }
}
```

Figura 45. Petición PUT.

En este caso realizamos una petición PUT para actualizar el CB y el programa en la Rasberry que consulta continuamente el valor de este campo, actualiza el valor de su variable "cerradura" simulando como si se abriera una cerradura real.

Con el método SetRequestHeader se establecen las cabeceras de la Petición HTTP, que en nuestro caso son las comentadas en el apartado 4.2 Context Broker.

A continuación, se muestra la petición comentada obtenida desde el programa Wireshark, donde se han señalado las características más importantes, Figura 46.

```
3653 253.451277    192.168.1.106        192.168.1.110        HTTP      521 [TCP Spurious Retransmission] PUT /v2/entities/Tienda003/attrs/terrario1 HTTP/1.
Frame 3653: 521 bytes on wire (4168 bits), 521 bytes captured (4168 bits) on interface 0
Ethernet II, Src: LgElectr_ff:ad:71 (cc:fa:00:ff:ad:71), Dst: Tp-LinkT_72:d2:6a (50:3e:aa:72:d2:6a)
Internet Protocol Version 4, Src: 192.168.1.106, Dst: 192.168.1.110
Transmission Control Protocol, Src Port: 60672, Dst Port: 1026, Seq: 1, Ack: 1, Len: 455
Hypertext Transfer Protocol
  PUT /v2/entities/Tienda003/attrs/terrario1 HTTP/1.1\r\n
  Expect: 100-continue\r\n
  Content-Type: application/json\r\n
  Fiware-ServicePath: /\r\n
  X-Unity-Version: 2019.1.0f2\r\n
  Fiware-Service: openiot\r\n
  Content-Length: 101\r\n
  User-Agent: Dalvik/2.1.0 (Linux; U; Android 6.0.1; Nexus 5 Build/M4B30Z)\r\n
  Host: 192.168.1.110:1026\r\n
  Connection: Keep-Alive\r\n
  Accept-Encoding: gzip\r\n
  \r\n
  [Full request URI: http://192.168.1.110:1026/v2/entities/Tienda003/attrs/terrario1]
  [HTTP request 3/3]
  File Data: 101 bytes
JavaScript Object Notation: application/json
  Object
    Member Key: type
    Member Key: value
      Object
        Member Key: temperatura
        Member Key: humedad
        Member Key: cerradura
            String value: abierta
            Key: cerradura
```

Figura 46. Captura Wireshark PUT.

Con la explicación de la APP que se ha llevado a cabo termina la parte de desarrollo. Continuaremos con las Conclusiones y las Líneas Futuras de este Proyecto.

7 CONCLUSIONES Y LÍNEAS FUTURAS

No creo que haya alguna emoción más intensa para un inventor que ver algunas de sus creaciones funcionado.

- Nikola Tesla -

Con la realización de este proyecto hemos logrado un sistema que permite gestionar los hábitats de los animales de manera remota. De esta forma, podemos monitorizar los parámetros más importantes y controlar algún actuador, como, por ejemplo, la cerradura para animales peligrosos. Todo esto ha sido posible gracias a las herramientas anteriormente explicadas (IoT, Servicios Web, Andoid, etc).

El proyecto que se ha tratado en este documento no es más que una primera versión académica de una posible herramienta de gestión para este tipo de negocio. Este trabajo no está preparado para el correcto funcionamiento de esta plataforma en un entorno real. Necesita algunas mejoras relacionadas con la seguridad, la eficiencia, recursos … A continuación, se comentará las posibles mejoras que, a mi parecer, considero importante realizar en este trabajo para que, en un futuro, sea posible su implementación en un establecimiento.

Un punto con especial importancia y no tratado en este trabajo ha sido la Seguridad Informática. En nuestro caso, cualquier persona que conozca la IP tiene acceso al Servidor o al Context Broker, por lo que podría interactuar y efectuar cambios sobre ellos si quisiera. Este problema se solventaría con un proceso de autenticación en el servidor, de forma que el usuario tenga una cuenta (usuario y contraseña) asociada. Así solo podrá acceder a las tiendas para las que tenga autorización. Espero poder completar este proyecto y suplir esta carencia el próximo año ayudándome de los conocimientos obtenidos en el Máster de Ciberseguridad de la Universidad de Sevilla.

Otro aspecto relativo a la seguridad es que las peticiones HTTP no van cifradas (http), pudiéndose cifrar estas comunicaciones para obtener una mayor confidencialidad de los datos.

Asimismo, nuestro servidor se encuentra alojado en una red privada sin acceso desde el exterior. De esta forma se simplifica el escenario y además nos protegemos de la posibilidad de que otro usuario pueda acceder a nuestro proyecto y realizar cambios con fines maliciosos. Si se deseara salir al exterior, es suficiente con añadir un DNAT en el router para que cambie la dirección pública de destino a la dirección privada del servidor. A pesar de esto considero que la opción más interesante y que posiblemente implemente en un futuro será la de alojar este servicio en la nube (clouding) para mejorar su accesibilidad.

En cuanto a la cerradura para animales peligrosos que se ha implementado mediante software, creo que podría ser una mejora de la parte de electrónica de este proyecto. Habría que conectarla a la Raspberry y hacer que cuando reciba la respectiva orden “ABRIR” de la aplicación móvil, ésta se abra. No se ha podido desarrollar en este proyecto por factores económicos, concretamente, debido al alto precio de este tipo de componentes.

Otra carencia detectada en este trabajo es que, en nuestra implementación, solo hemos tenido en cuenta dos hábitats con sensores instalados (temperatura / ph) y otros dos terrarios de forma simulada. En un entorno real,

deberían colocarse en todos los terrarios y/o acuarios que lo necesiten. Además, nosotros solo hemos desarrollado una tienda, pero un Gestor puede tener asociadas varias tiendas a la vez. Añadiendo los componentes que faltan y con una breve modificación en el código de control de la Raspberry y en el Servidor Web, sería posible aumentar el número de hábitats y ampliar el sistema.

Como aspecto positivo cabe destacar la fácil adaptabilidad a cambios que ofrece el sistema. Comentaremos esta característica con el siguiente ejemplo real que ocurrió durante el transcurso de la realización del proyecto:

Una vez realizado el desarrollo de una primera versión de todo el sistema, decidí ponerlo a funcionar en un entorno real e interactuar con un profesional en animales exóticos para detectar posibles mejoras de los aspectos más importantes que él necesitaba monitorizar.

Tras el estudio realizado con el profesional, llegué a la conclusión de que los tres parámetros más importantes y fáciles de implementar fueron:

- Humedad y temperatura: la monitorización de estas magnitudes es tan importante debido a que los calentadores y humidificadores, que se encargan de mantenerlas en unos determinados valores constantes, suelen dejar de funcionar. Según la experiencia del encargado, este tipo de fallos suelen suceder con una periodicidad de un componente por cada mes, lo cual puede ocasionar problemas catastróficos si no se reemplazan a tiempo.
- PH, en un principio desconocía la importancia de monitorizar este parámetro en el mundo animal. Durante la reunión se profundizó sobre el trabajo que el Gestor realizaba, y entre ellos estaba obtener medidas de PH de los acuarios. Usaba un mismo medidor electrónico para todos. En su explicación detalló que el ph y el cloro son vitales para los peces y necesita estar siempre en un determinado valor, y para ello se añaden determinados líquidos que ajustan estos niveles. En la siguiente referencia observamos un ejemplo de estos líquidos [38].

  El Gestor realiza dos medidas diarias a cada acuario, con un tiempo mínimo de 2 minutos. Ello es debido a que tiene que tomar la medida, limpiar el sensor con agua destilada y secarlo. A raíz de esto decidí implementar un mecanismo que permitiera al gestor comprobar estos niveles de una forma más rápida, sencilla y cómoda. En este caso particular, dispone de 29 acuarios, a 2 minutos por cada uno, dedica 58 minutos cada vez que comprueba y mide estos niveles. Con la aplicación desarrollada, podría comprobar todos estos parámetros con un clic en menos de un minuto. Además, el medidor de PH que hemos usado es muy económico y efectivo.

A continuación, mostramos el lugar que fue visitado para obtener la información de nuestro estudio, Figura 47.

Figura 47. Terrarios / Acuarios.

También se muestra la implementación realizada en dos de estos hábitats reales, Figura 48.

Figura 48. Prueba de implantación real.

La instalación ha sido realizada con la supervisión de un profesional para que no se vea alterado el bienestar de los animales. En ningún momento se ha manipulado a ninguno, siendo el objetivo final de este trabajo ayudar a mejorar la vida de las mascotas en hábitats que se adapten a sus necesidades.

En este Proyecto nos hemos dedicado a la creación de un sistema completo para la gestión y monitorización basado en el IoT, centrándonos en concreto en los hábitats de los animales. Pero el proyecto desarrollado es fácilmente implementable en una multitud de aplicaciones distintas como puede ser la domótica de las casas, plantaciones agrícolas, granjas de apicultores … Cambiando los tipos de sensores que obtienen la información

que se va a monitorizar y modificando brevemente el *daemon* ejecutado en la Raspberry, se abre el abanico de posibles implementaciones para este trabajo.

Para finalizar, destacar que la complejidad de este trabajo no ha consistido en la realización de una tarea específica como puede ser la interconexión de sensores, o bien el desarrollo de alguna de las aplicaciones. La dificultad ha recaído en la comunicación de todas estas herramientas gratuitas y de código abierto para conseguir un trabajo común, interconectado y además que sea útil en un ámbito comercial. Desde el punto de vista del Trabajo Fin de Grado me parece una perfecta forma de plasmar algunos de los conocimientos más importantes adquiridos en la carrera. Entre ellos consta la electrónica, útil para la interconexión de los sensores con el microprocesador, así como con el conversor. También todo lo relativo a la programación necesaria para el desarrollo de la aplicación en Python, en Android y el Servidor Web. Por último, el conocimiento de sistemas distribuidos y redes para la correcta comunicación entre todos los distintos componentes de la plataforma.

ANEXO A: INSTALACIÓN Y CONFIGURACIÓN DE CONTEXT BROKER, CYGNUS Y MYSQL CON DOCKER

A continuación, se detallarán los pasos necesarios para la instalación de Orion Context Broker. Para instalar esta herramienta se ha utilizado la tecnología basada en contenedores Docker. Además, para esta instalación se ha usado la Máquina Virtual comentada anteriormente en el apartado 2.2.1 con Ubuntu 16.04.

Concretamente hemos usado Docker-Compose como una herramienta para levantar múltiples contenedores, usando esta misma para el Context-Broker, Cygnus y la Base de Datos de MySQL. En el siguiente Anexo se explicará la parte correspondiente de Docker-Compose para cada una de estas herramientas.

A.1 Instalación de Docker

Para realizar la instalación se debe ejecutar los siguientes comandos en la terminal:

```
$ sudo apt-get update
$ sudo apt-get install apt-transport-https ca-certificates curl
software-properties-common

$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-
key add -


$ sudo apt-get install docker-ce
```

Se puede comprobar la instalación con el siguiente comando:

```
$ docker --version
```

Con el que obtendremos la Figura 49:

```
 Git commit:        74b1e89
 Built:             Thu Jul 25 21:21:05 2019
 OS/Arch:           linux/amd64
 Experimental:      false

Server: Docker Engine - Community
 Engine:
  Version:          19.03.1
  API version:      1.40 (minimum version 1.12)
  Go version:       go1.12.5
  Git commit:       74b1e89
  Built:            Thu Jul 25 21:19:41 2019
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.2.6
  GitCommit:        894b81a4b802e4eb2a91d1ce216b8817763c29fb
 runc:
  Version:          1.0.0-rc8
  GitCommit:        425e105d5a03fabd737a126ad93d62a9eeede87f
 docker-init:
  Version:          0.18.0
  GitCommit:        fec3683
```

Figura 49. Versión de Docker.

En Segundo lugar, debemos instalar la herramienta Docker-Compose.

```
$ sudo curl -L
$ https://github.com/docker/compose/releases/download/1.22.0/docker-
compose-$(uname -s)-$(uname -m) -o /usr/local/bin/docker-compose

$ sudo chmod +x /usr/local/bin/docker-compose
```

De igual forma podemos ejecutar:

```
$ docker-compose --version
```

Al ejecutar en nuestro PC se obtiene la Figura 50:

```
docker-compose version 1.22.0, build f46880fe
```

Figura 50.Versión de Docker-Compose

Por último, podemos crear un grupo Docker y se agrega el usuario actual para poder ejecutar los comandos sin usar el usuario root. Este paso no es necesario para este proyecto, pero se considera una buena práctica.

```
$ sudo groupadd Docker
$ sudo usermod -aG docker $USER
```

Para realizar esta tarea de instalación de Docker Compose se ha seguido principalmente la Guía de Fiware[19].

Existe un comando para levantar un contenedor de prueba, para comprobar si todo se ha instalado correctamente y poder seguir con la instalación del resto de herramientas.

```
$ docker run hello-word
```

A.2 Docker-Compose. Instalación de Context-Broker

Tras instalar Docker, utilizamos un archivo en formato "*yaml"* para desplegar los contenedores, entre ellos el Context Broker.

Se ha creado un directorio "Docker" en el que se incluye el fichero "*docker-compose.yml"* que contiene lo relativo a todos los servicios instalados en contenedores. En este apartado se va a tratar solo la parte del Context-Broker y de la base de datos MongoDB. En el siguiente apartado continúa la explicación de este documento para el resto de servicios.

```
version: "3"
services:
  mongodb:
    image: mongo:3.4.2
    hostname: mongodb
    container_name: mongodb
    expose:
      - "27017"
    ports:
      - "27017:27017"
    volumes:
      - mongodb-volume:/data
    command: mongod --smallfiles

  orion:
    image: fiware/orion:1.7.0
    hostname: orion
    container_name: orion
    links:
      - mongodb
    expose:
      - "1026"
    ports:
      - "1026:1026"
    command: -dbhost mongodb
```

Fragmento I de Docker-compose.yml

En el fichero *Docker-compose.yml* aparecen los siguientes apartados:

- La versión de docker compose que se utilizará, en nuestro caso se ha utilizado la versión 3.
- La etiqueta "*service"* que describe cada uno de los contenedores que se crearán y para que servicio en concreto. Dentro de esta etiqueta se han incluido Orion Context Broker, MongoDB, Cygnus y MySQL. Estas dos últimas no aparecen en el Fragmento I, pero se comentará posteriormente. A continuación, se detallan las etiquetas comunes a la mayoría de los servicios desplegados:
    - *image*: esta etiqueta indica el nombre de la imagen a partir del cual se crea el contenedor.
    - *hostname:* especifica el nombre del host.
    - *container_name*: especifica el nombre que tendrá el contenedor.
    - *command*: permite ejecutar comandos de la consola tras la creación del contenedor.
    - *evironment*: se utiliza para crear variables de entorno para los contenedores desplegados.
    - *links/depends_on*: indica si un contenedor o servicio depende de otro para poder ejecutarlo.
- *volumes*: con esta etiqueta se consigue que el directorio actual se mapee directamente con el lugar donde hemos creado la aplicación.

Centrándonos en Orion y MongoDB observamos que el primero se mapea al puerto 1026 mientras que la base de datos al 27017. Es importante destacar que en la etiqueta "*command"* de Orion se realiza una llamada a "*dbhost -mongodb*" para indicar el nombre del servidor de base de datos al que el Context Broker se conectará. Con la etiqueta "*links"* indica que el contenedor con el OCB depende del contenedor mongo.

A.3 Instalación Cygnus y MySQL

Tanto Cygnus como MySQL se han levantado en dos Dockers, para ello se ha usado la herramienta Docker Compose y el fichero *"docker-compose.yml"*. A continuación, mostramos el otro fragmento de este fichero donde se levanta los contenedores de estas herramientas.

```
mysql:
  image: mysql:latest
  hostname: mysql
  container_name: mysql
  expose:
    -"3306"
  ports:
    - "3306:3306"
  environment:
    -MYSQL_ROOT_PASSWORD=mysql

cygnus:
  image: fiware/cygnus-ngsi:latest
  hostname: cygnus
  container_name: cygnus
  links:
    -mysql:iot-mysql
  expose:
    -"5050"
  ports:
    - "5050:5050"
    - "8081:8081"
  environment:
    -CYGNUS_MSQL_USER=root
    -CYGNUS_MYSQL_PASS=mysql
```

Fragmento II de Docker Compose.

Se puede observar en el anterior fragmento que Cygnus utiliza el puerto 5050 mientras que MySQL el 3306. También vemos que para que Cygnus sea ejecutado primero debe de existir el docker MySQL ("`links:-mysql:iot-mysql`"). Las variables que aparecen en *enviroment* hace referencia al usuario y contraseña de la base de datos MySQL.

Igual que en el apartado anterior con el comando:

```
$ docker-compose -f docker-compose-demo.yml up -d
```

Se crean todos los contenedores con las imágenes (si no han sido instaladas previamente) de las herramientas. Esto permite a Cygnus crear y actualizar la Base de Datos a la que accederá el Servidor Web.

En nuestro Proyecto en particular tenemos todos los contenedores en la misma máquina, por lo que estos están accesibles a través de los puertos mostrados y de la dirección de la interfaz de la máquina virtual, en nuestro caso usaremos la ip privada 192.168.1.110.

A.4 Docker-Compose. Ejecución de servicios.

Para desplegar los contenedores y ejecutar los servicios basta con ejecutar el siguiente comando:

```
$ docker-compose -f docker-compose-demo.yml up -d
```

Podemos comprobar los servicios desplegados con:

```
$ docker ps
```

Obteniendo la salida mostrada en la Figura 51.

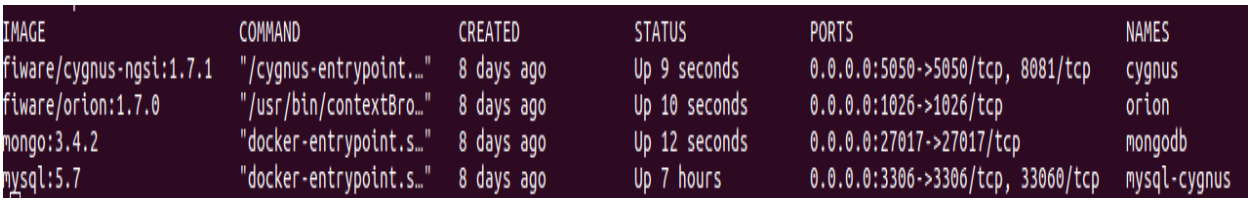

IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
fiware/cygnus-ngsi:1.7.1	"/cygnus-entrypoint.…"	8 days ago	Up 9 seconds	0.0.0.0:5050->5050/tcp, 8081/tcp	cygnus
fiware/orion:1.7.0	"/usr/bin/contextBro…"	8 days ago	Up 10 seconds	0.0.0.0:1026->1026/tcp	orion
mongo:3.4.2	"docker-entrypoint.s…"	8 days ago	Up 12 seconds	0.0.0.0:27017->27017/tcp	mongodb
mysql:5.7	"docker-entrypoint.s…"	8 days ago	Up 7 hours	0.0.0.0:3306->3306/tcp, 33060/tcp	mysql-cygnus

Figura 51. Docker-ps.

Al igual que para levantarlos se utiliza el mismo comando para detener la ejecución de los contenedores y parar los servicios.

```
$ docker-compose -f docker-compose-demo.yml down -d
```

Para borrar totalmente los contenedores, con todos sus datos y las imágenes de los servicios se utiliza estos dos últimos comandos:

```
$ docker rm $(docker ps -aq)
$ docker rmi $(docker images -aq)
```

A.5 Instalación MySQL WorkBench

Para instalar esta herramienta hemos usado el gestor de paquetes APT, este se puede instalar con los siguientes comandos:

```
$ sudo apt-get update
$ sudo apt-get install mysql-workbench
```

ANEXO B: CONFIGURACIÓN Y CONEXIÓN DE SENSORES CON RASPBERRY PI

En esta parte del trabajo se detallará la conexión entre los sensores y la Raspberry (Figura 52). En primer lugar, veremos la conexión del sensor DTH-11. A continuación comentaremos la conexión entre el medidor de PH con el conversor ADC y este con la Raspberry.

También se centra este apartado en los pasos seguidos para la instalación de las librerías necesarias para interpretar los datos almacenados por los sensores.

La versión y los detalles tanto del software como de la Raspberry Pi se encuentran en el apartado 2.1 *Recursos Software.*

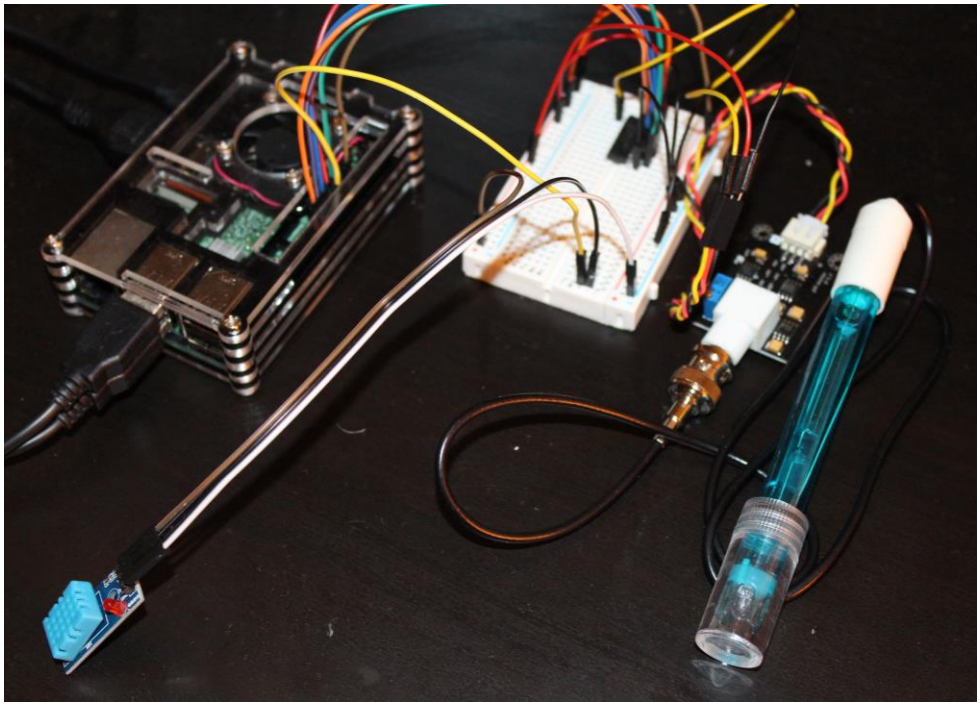

Figura 52. Interconexión de sensores I.

B.1 Instalación Python

Para que sea posible interpretar los datos captados por los sensores y enviar estos al Context Broker se ha usado el programa desarrollado en Python *daemon_sensor.py.* Por lo que es requisito fundamental instalar Python en la Raspberry. Hemos usado el gestor de paquetes APT, que es el gestor por defecto en sistemas basados en Debian. Para ello hemos usado los siguientes comandos:

- Actualizamos el gestor de paquetes necesario para la instalación.

```
$ sudo apt-get update -y
$ sudo apt-get upgrade -y
```

- Instalamos la última versión de Python.

```
$ sudo apt-get install python3-pip
```

- Comprobamos la versión instalada.

```
$ sudo python --version
```

B.2 Instalación de librerías

En este apartado se explica los pasos a seguir para instalar las librerías necesarias para los dos sensores usados en el proyecto.

- Instalamos git, para descargar contenido de GitHub.

```
$ sudo apt-get install git-core
```

- Descargamos el repositorio git de la librería para el sensor DTH11

```
$ git clone https://github.com/adafruit/Adafruit_Python_DHT.git
```

- Entramos en la carpeta descargada

```
$ cd Adafruit_Python_DHT
```

- Instalamos la librería con Phyton

```
$ sudo python setup.py install
```

- Instalamos las librerías para el conversor MCP3008, en este caso se ha usado otro método de instalación debido a que la instalación por paquetes producía un error para esta librería concreta.

```
$ sudo pip3 install adafruit-blinka
$ sudo pip3 install adafruit-circuitpython-mcp3xxx
```

B.3 Conexión sensor DHT11

La interconexión de este sensor es muy sencilla ya que va directamente conectado a los pines GPIOs de la Rapsberry. Usaremos para referirnos a los pines la nomenclatura de la Figura 53:

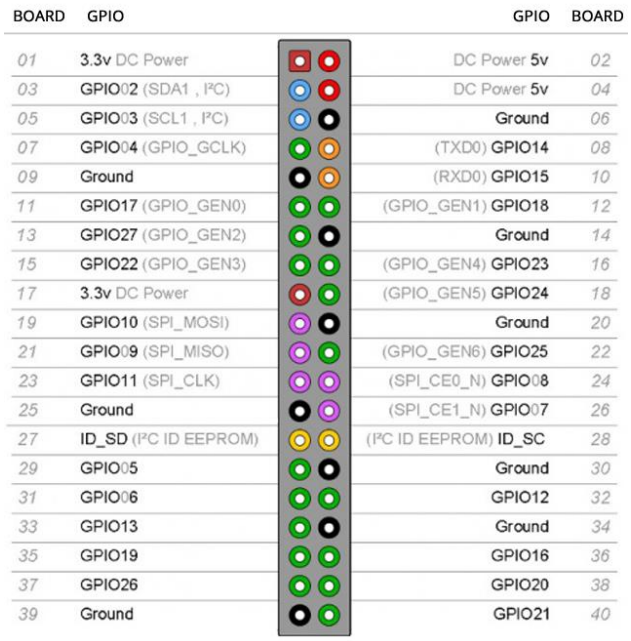

Figura 53.Pines GPIOs [27]

Para este sensor solo necesitamos las tres conexiones básicas.

DHT11	Raspberry Pi
VCC	Pin 1 (3.3V)
TX	Pin 16 (GIPIO23)
GND	Pin 6 (Ground)

Tabla 4.Conexión sensor DHT-11.

Para simplificar la conexión, no se conectan directamente sobre los pines de la Raspberry, sino que se ha usado una protoboard que va conectada a los pines y sobre esta los sensores.

B.4 Conexión del medidor de PH

Debido a que la Raspberry Pi no tiene forma de leer entradas analógicas, al contrario que los microcontroladores Arduino. Se ha tenido que instalar en el circuito un conversor MCP3008 para realizar esta conversión y poder obtener el PH del agua. Para todo este apartado se ha usado como guía el documento de Adafruit disponible en la siguiente referencia [28].

En primer lugar, se muestra un esquema de los pines del conversor, Figura 54.

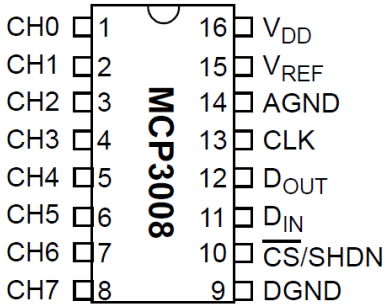

Figura 54. MCP3008 pines.

En la siguiente tabla se presenta la conexión Conversor-Raspberry.

MCP3008	Raspberry Pi
VDD	Pin 1 (3.3V)
VREF	Pin 1 (3.3V)
AGND	Pin 6 (Ground)
CLK	Pin 23 (SPI_CLK)
Dout	Pin 21 (MISO)
Din	Pin 19 (MOSI)
CS/SHDN	Pin 15 (GPIO22)
DGND	Pin 6 (Ground)

Tabla 5. Conexión sensor de PH a conversor.

Hay que tener habilitado el puerto I2C de la Raspberry Pi que por defecto suele venir *"Disable"*. Para realizar eso se debe introducir el comando que se muestra a continuación y activar el *"P5 I2C"* del menú *"Raspberry Pi Software Configuration Tool"*.

```
$ sudo raspi-config
```

Por último, el sensor de PH va conectado al CH0 (puede conectarse a cualquier canal del conversor). Además, debe ir alimentado a 3.3V (Pin 1) y conectado a tierra (Pin 6). Una vez realizada las conexiones con los pines de la Raspberry necesitamos conectar el Electrodo medidor de PH al controlador PhMeterv1.1 a través del BNC como se muestra en la Figura 55.

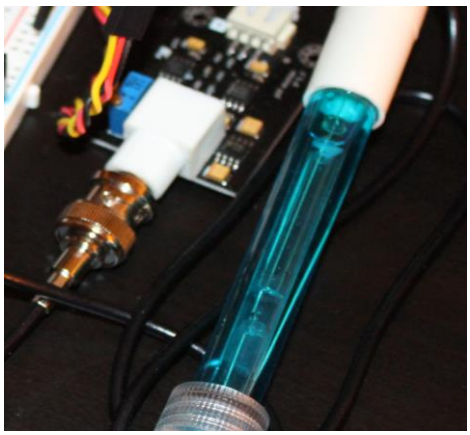

Figura 55. Conexión BNC.

ANEXO C: INSTALACIÓN Y CONFIGURACIÓN DE UNITY

Este programa para el desarrollo de aplicaciones móviles, se ha instalado en el sistema operativo Windows 10 del ordenador. Esto es debido a que no está disponible para Linux.

C.1 Instalación Unity Hub

Para descargar Unity3D hemos usado el asistente de instalación de esta plataforma, Unity Hub. Este se puede encontrar en la siguiente referencia [29].

En la Figura 56 se muestra el primer paso de la descarga.

Figura 56. Descarga Unity Hub.

Tras la descarga ejecutamos el "*.exe*" tras elegir la carpeta donde llevar a cabo la instalación y aceptar los términos y condiciones comenzará la instalación del Software.

C.2 Instalación Unity

Una vez tengamos instalado el asistente entonces procederemos a instalar la versión más reciente de Unity, para ello usaremos la siguiente página [30]. En la Figura 57 podemos observar este paso.

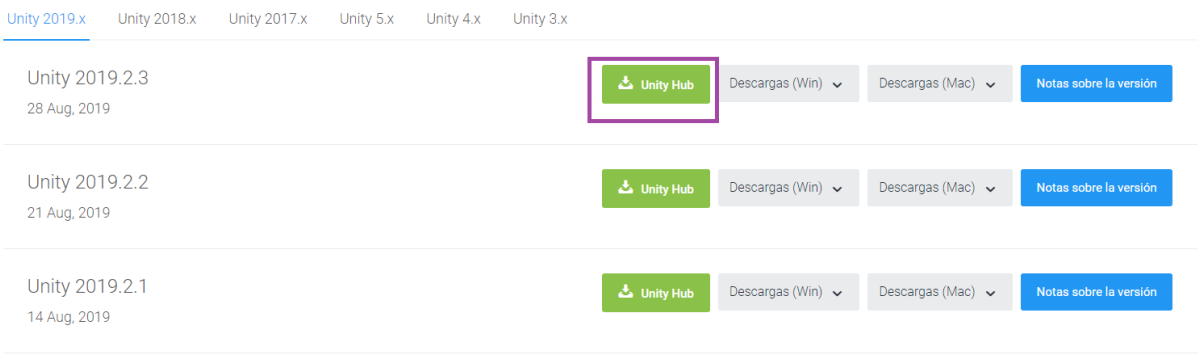

Figura 57. Descarga Unity I.

Una vez descargada la versión que necesitemos abrimos Unity Hub y seguimos los pasos mostrados en las siguientes Figuras 58, 59 y 60:

- Primero pulsamos en el botón ADD para añadir una versión de Unity como muestra la Figura 58.

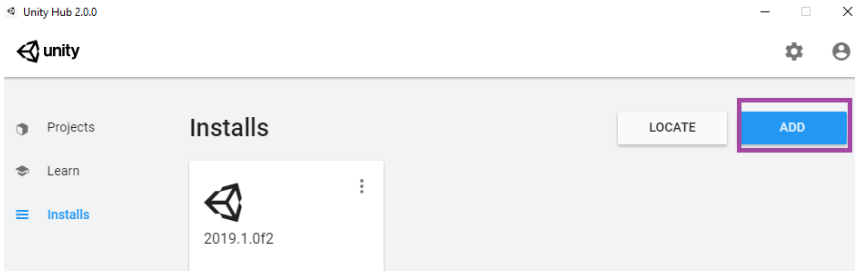

Figura 58. Descarga Unity II.

- Añadimos la versión descargada anteriormente, Figura 59.

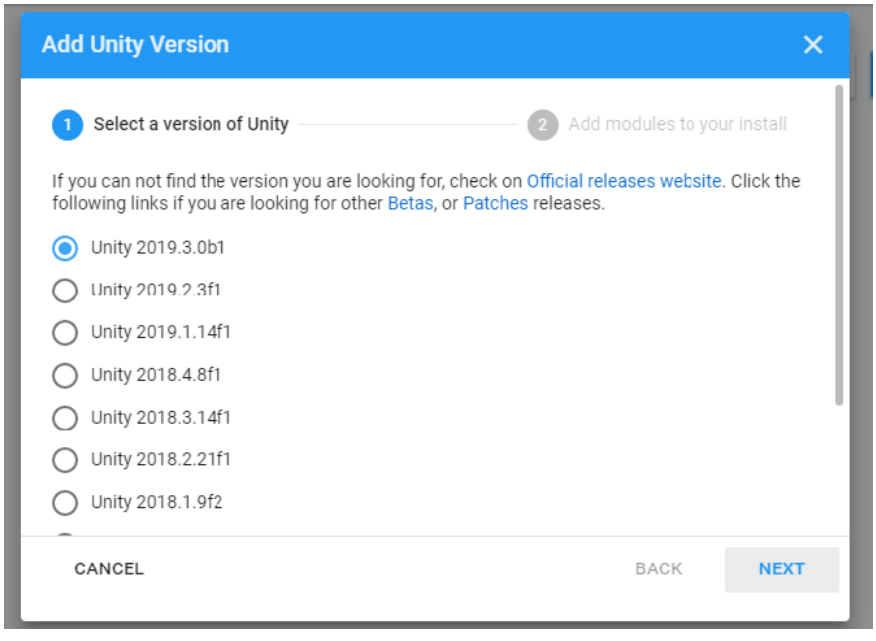

Figura 59. Descarga Unity III.

- Seleccionamos los módulos que necesitemos en nuestro caso es importante seleccionar *"Android Build Support"* como muestra la siguiente Figura 60.

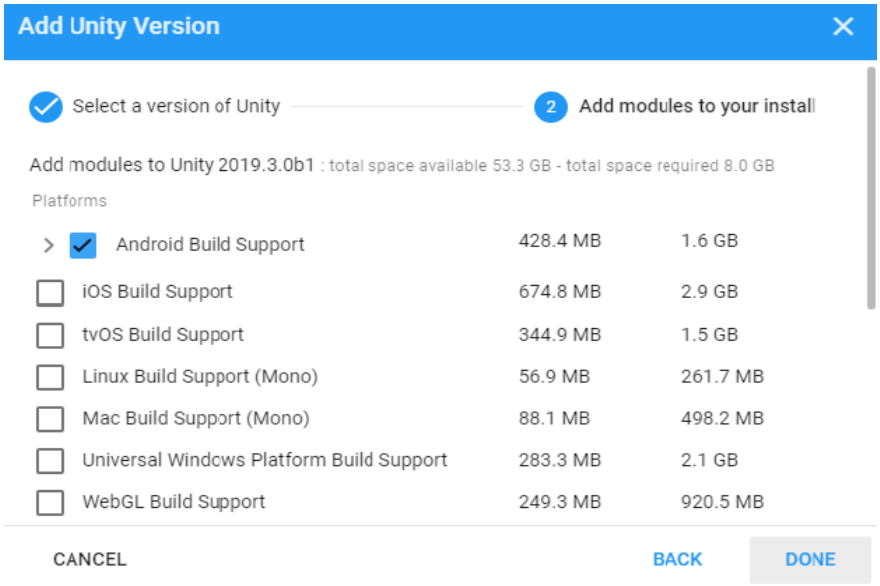

Figura 60. Descarga Unity IV.

El módulo seleccionado es de vital importancia para el desarrollo de nuestra App ya que este instalará los siguientes componentes:

- SDK: El Kit de Desarrollo Software es un conjunto de herramientas para crear nuestra aplicación software.
- JDK: Kit de Desarrollo Java es un entorno de desarrollo para crear aplicaciones, applets y componentes utilizando el lenguaje de programación Java. Mas información en la siguiente referencia [31].
- NDK: es un conjunto de herramientas que te permite desarrollar Apps Android mediante lenguajes como C y C++ [32].

C.3 Creación de un nuevo Proyecto Unity

Una vez que se ha instalado el Software anterior lo siguiente es crear el Proyecto para empezar a desarrollar la app para ello solo tenemos que acceder a la Ventana *"Projects"* del Unity Hub y añadir el nuevo Proyecto como muestra en las siguientes figuras.

- Primero seleccionamos *"New"* para crear el nuevo Proyecto, Figura 61.

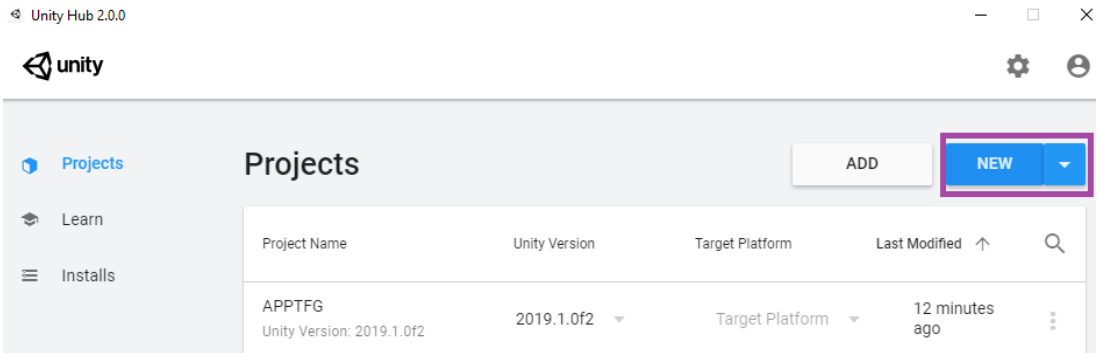

Figura 61. Creación Proyecto Unity I.

- A continuación, seleccionamos el tipo de proyecto que vamos a realizar. En nuestro caso será un proyecto 3D como muestra en la Figura 62.

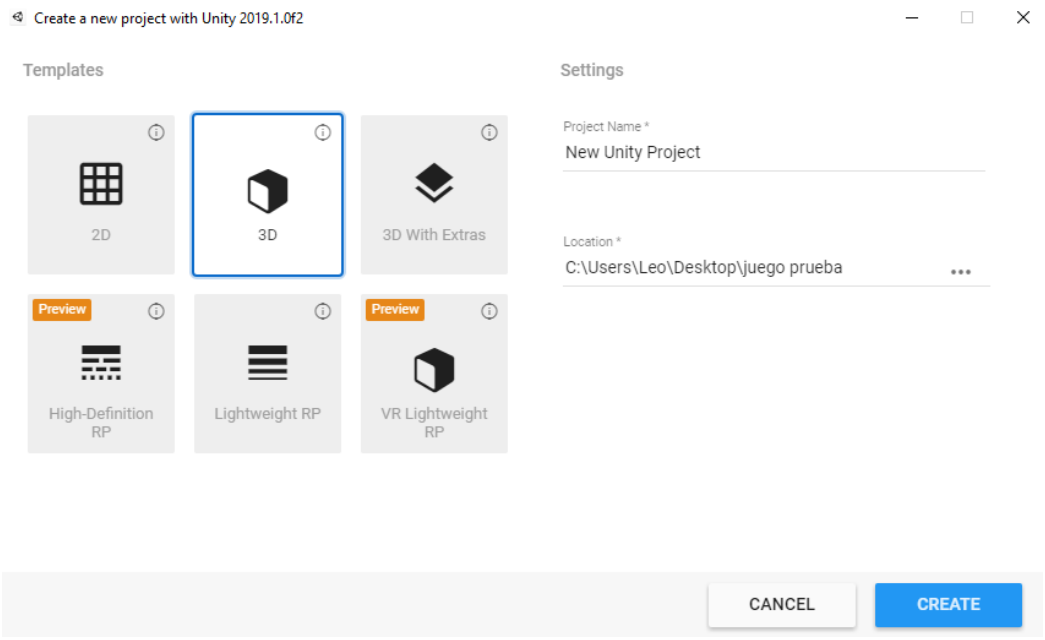

Figura 62. Creación Proyecto Unity II.

Con los pasos anteriores se completa la instalación de Unity y la creación del proyecto. Quedando configurado para comenzar el desarrollo de la App "*Control Yours Animals"*. Para más información sobre la plataforma Unity consultar la referencia [33].

ANEXO D: INSTALACIÓN Y CONFIGURACIÓN DE STS

En este apartado se explica la instalación de Spring Tools Suite versión 4 así como la herramienta Hibernate Tools. Esta instalación se llevará a cabo en la máquina virtual con sistema operativo Ubuntu. Para escribir este Anexo se ha seguido la guía de la referencia [34].

D.1 Instalación de STS

1. Comprobamos la versión del JDK que tenemos instalado con el siguiente comando.

```
$ java -version
```

Si no lo tuviéramos instalado debemos ejecutar los comandos:

```
$ sudo add-apt-repository ppa:webupd8team/java
$ sudo apt update
$ sudo apt install oracle-java8-installer
```

2. Instalamos Maven, para más información consultar la siguiente referencia [35].

```
$ sudo apt install maven
```

3. Descargamos Spring Tool Suite desde la página oficial en la siguiente dirección [36]. Seleccionamos la opción marcada en la Figura 63.

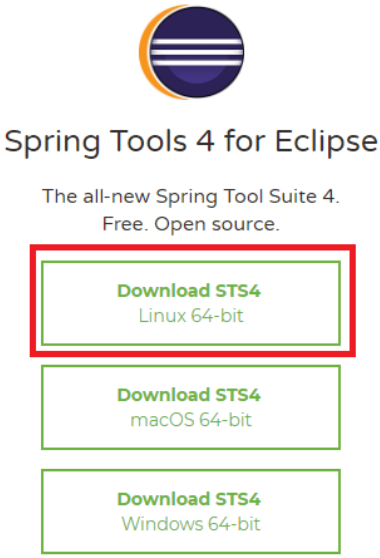

Figura 63. Instalación Spring Tool 4.

4. A continuación, accedemos a la carpeta de descargas lo movemos a la carpeta */opt*, descomprimimos el *.tar* y creamos un enlace simbólico para acceder a este.

```
$ cd /Descargas
$ sudo mv spring-tool-suite-4-4.3.2.RELEASE-e4.12.0-linux.gtk.x86_64.tar.gz / opt
$ cd /opt
$ cd tar zxvf spring-tool-suite-4-4.3.2.RELEASE-e4.12.0-linux.gtk.x86_64.tar.gz
$ sudo ln - s /opt/sts-4.3.2.RELEASE/SpringToolSuite4/usr/local/bin/sts
```

5. Creamos un launcher en la carpeta /usr/share/applications introduciendo el siguiente comando:

```
$ sudo nano /usr/share/applications/sts.desktop
```

Para finalizar añadimos el siguiente contenido al fichero:

```
[Desktop Entry]
Name=Spring Tool Suite 4
Comment=Sprint Tool Suite 4
Exec=/usr/local/bin/sts
Icon=/opt/sts-4.3.2.RELEASE/icon.xpm
StartupNotify=true
Terminal=false
Type=Application
Categories=IDE;Development;Java;
```

D.2 Creación del Proyecto STS

1. Creamos un nuevo proyecto Spring, indicando la versión, grupo, artefacto … como se indica en la Figura 64.

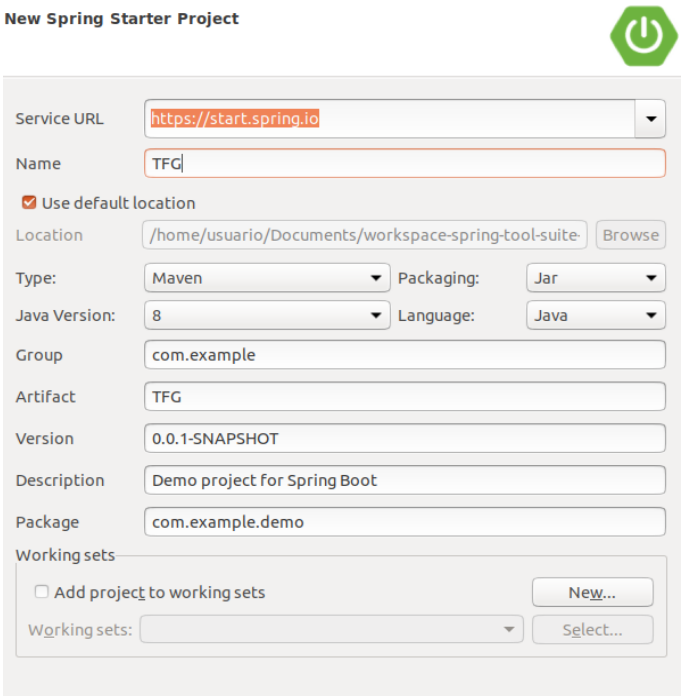

Figura 64. Creación de Proyecto Spring I.

2. Seleccionamos las herramientas que vamos a utilizar en nuestro proyecto en nuestro caso serían las marcadas en la Figura 65.

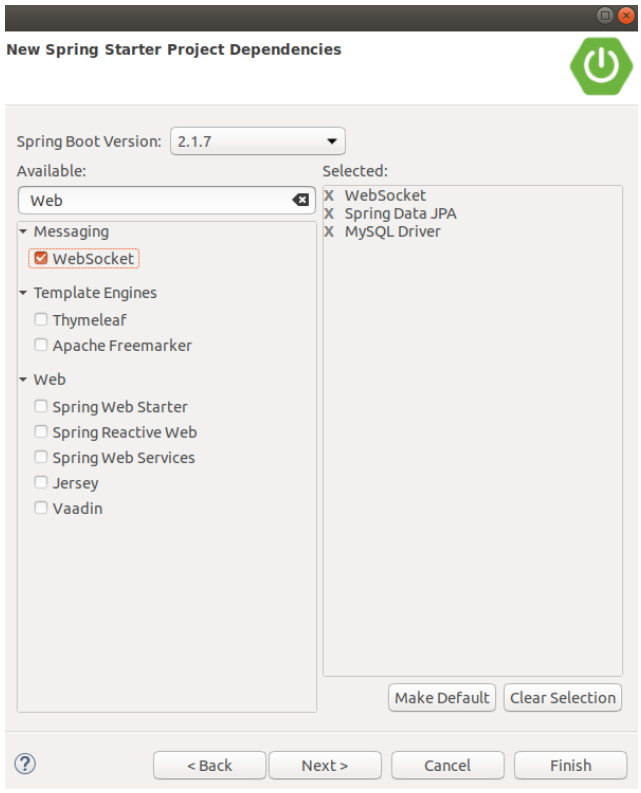

Figura 65.Creación de Proyecto Spring II.

Una vez realizado lo anterior ya tendríamos creado nuestro proyecto Spring con el sistema de directorios adecuado y los ficheros de configuración correspondientes.

ANEXO E: CONFIGURACIÓN DE LA MÁQUINA VIRTUAL

En este anexo se explica cómo debe de estar configurada las interfaces de acceso a Internet de la máquina virtual. En nuestro caso utilizaremos Virtual Box, pero el procedimiento es similar en el resto de máquinas virtuales.

E.1 Configuración de ajustes de Red

En primer lugar, debemos tener instalada la máquina virtual con la imagen de Ubuntu, para realizar esto se puede consultar la guía de la escuela de la siguiente referencia [37]

Para configurar el acceso a internet y a la subred privada debemos seguir los siguientes pasos:

1. Seleccionamos la configuración de la máquina específica.
2. Accedemos a la pestaña de *"Red"*, comprobamos que el primer adaptador tiene los valores que aparecen en la Figura 66.

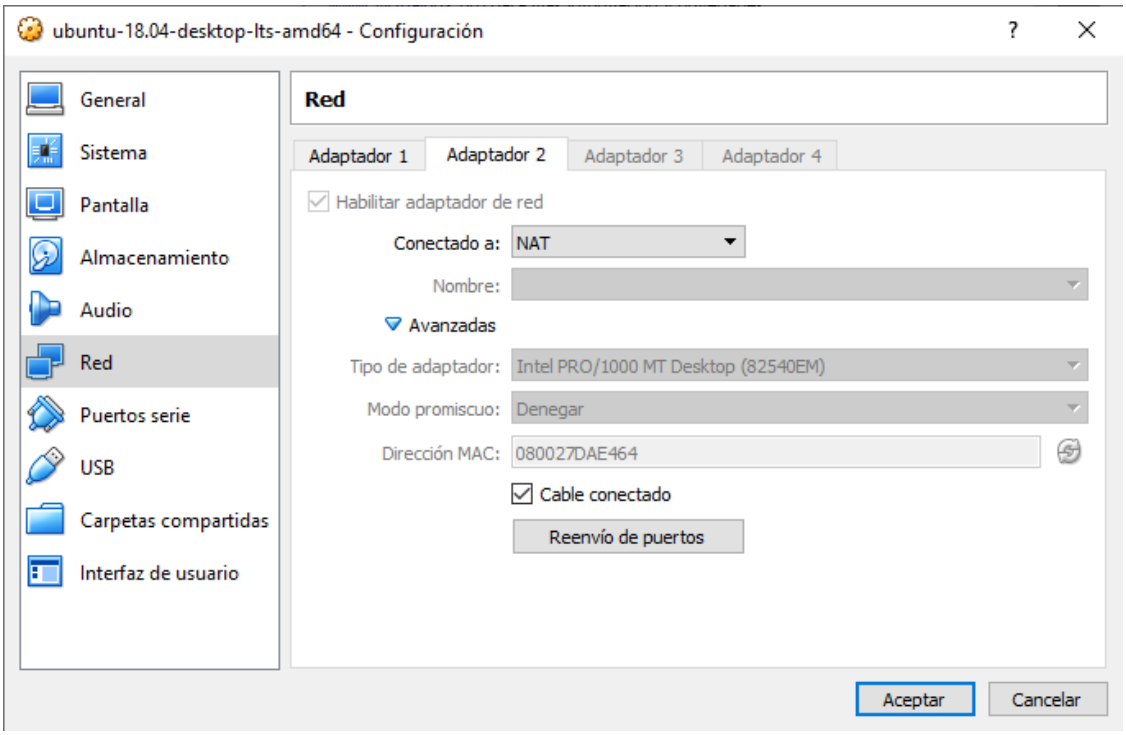

Figura 66. Configuración Máquina Virtual I.

3. Añadimos otro *"Adaptador de Red"* y lo configuramos como se muestra a continuación en la Figura 67.

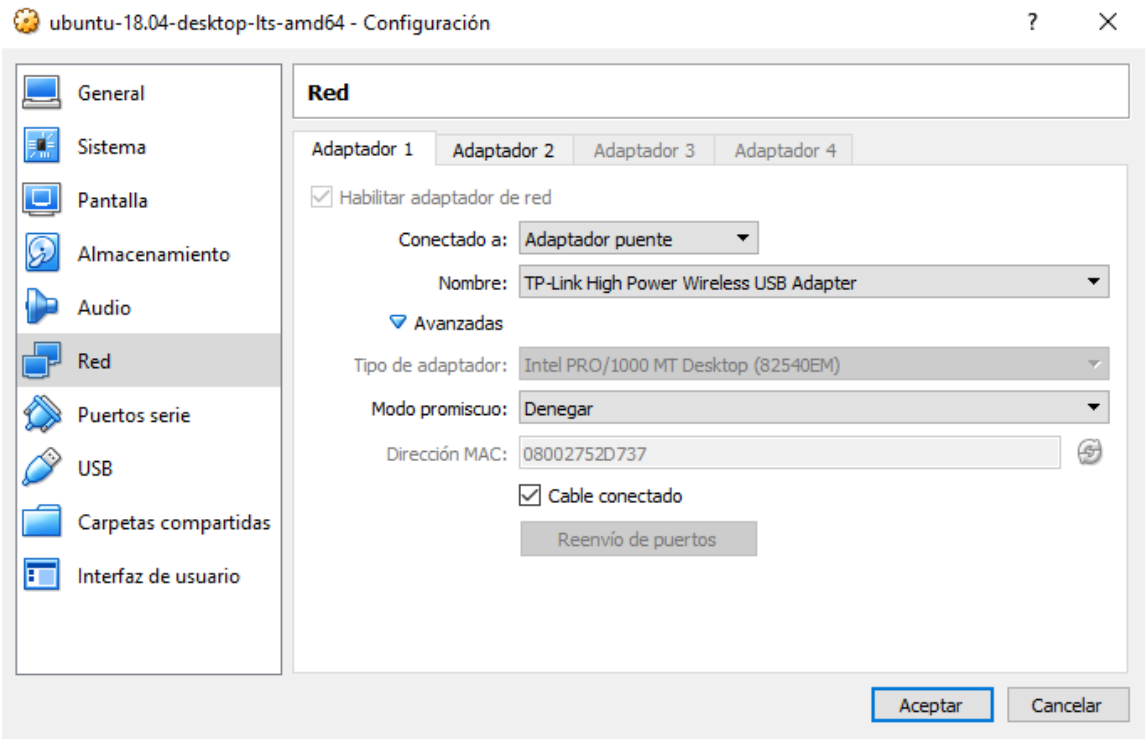

Figura 67. Configuración Máquina Virtual II.

De esta forma tendremos conexión a internet mediante el segundo *"Adaptador de Red"* que hará DNAT para salir a esta, mientras que con el *"Adaptador en Puente"* obtenemos una dirección IP de la red privada a la que está conectada la máquina. Así podemos referirnos a esta dirección IP para acceder a los servicios alojados en los contenedores, así como al Servidor Web.

ANEXO F: PUESTA EN MARCHA DEL SISTEMA

En este Anexo describiremos los pasos necesarios para poder poner en marcha todo el sistema que hemos desarrollado.

F.1 Sensores y Raspberry Pi

Para el correcto envío de datos por parte de la Raspberry se deben seguir los siguientes pasos:

1. Comprobar la correcta conexión de los sensores, el conversor y los pines GPIOs. Para realizar esta tarea consultar el Anexo C: Configuración y conexión de sensores con Raspberry Pi.
2. Incluir el código del programa *"daemon_sensores.py"* en un directorio.
3. Accedemos al directorio donde se encuentra el programa anterior y lo ejecutamos con el siguiente comando:

```
$ cd proyecto
$ sudo python3 daemon_sensores.py
```

F.2 Contenedores Dockers.

Describe los pasos para poner en marcha las herramientas siguientes: Context Broker, MongoDB, Cygnus y MySQL.

1. Abrir y configurar la máquina virtual, Anexo F: Configuración de la Máquina Virtual.
2. Instalar todas las herramientas necesarias como se muestra en los Anexos vistos anteriormente.
3. Incluir el documento *"docker-compose-demo.yml"* en un directorio accesible.
4. Ejecutar el siguiente comando para levantar los contenedores correctamente.

```
$ cd docker
$ sudo docker-compose -f docker-compose.yml up
```

Para levantar los contenedores usamos el archivo *.yml* y la herramienta Docker Compose. Para realizar la misma tarea, pero sin usar esta herramienta consultar la referencia [39], esto no se ha tratado en este TFG.

F.3 Servidor Web.

1. Abrir y configurar la máquina virtual, Anexo F: Configuración de la Máquina Virtual.
2. Instalar el Servidor Web, Anexo E: Instalación y configuración de STS.
3. Abrimos Spring Tools Suite 4.
4. Añadimos todo el código necesario para la ejecución, éste se encuentra en Github [42].
5. Ejecutamos el servidor como se muestra en la Figura 68.

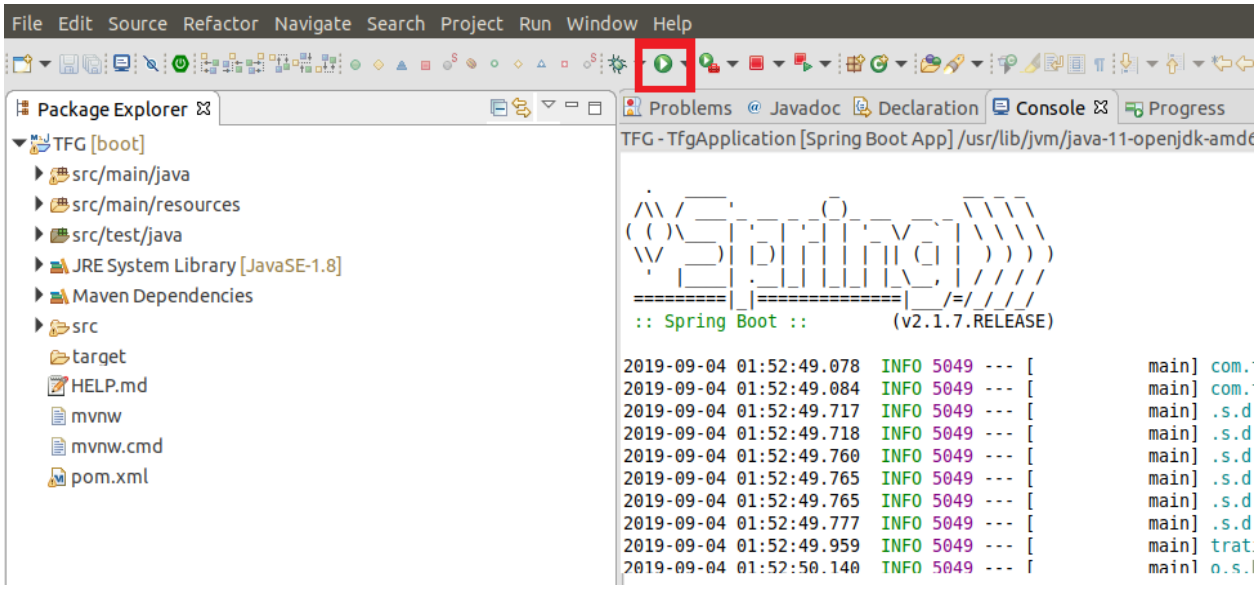

Figura 68. Ejecución Spring.

Una vez que hemos realizado todos los pasos anteriores podemos abrir la aplicación en nuestro teléfono móvil y gestionar los hábitats de los animales.

REFERENCIAS

[1] Adafruit Learning System, DHT11, DHT22 and AM2302 Sensors, [En línea]. Avaliable: https://cdn-learn.adafruit.com/downloads/pdf/dht.pdf .

[2] Wikipedia, Nexus 5, [En línea]. Avaliable: https://es.wikipedia.org/wiki/Nexus_5.

[3] MySQL, MySQL Workbench, [En línea]. Avaliable: https://www.mysql.com/products/workbench/

[4] David Marco, Integración de Eclipse y Spring Tool Suite, [En línea]. Avaliable: http://www.davidmarco.es/articulo/integracion-de-eclipse-y-springsource-tool-suite

[5] Postman, Logo de Postman (2018), [En línea]. Avaliable: https://www.getpostman.com/img/v2/media-kit/Logo/PNG/pm-logo-horiz.png

[6] Wikipedia, Unity, [En línea]. Avaliable: https://es.wikipedia.org/wiki/Unity_(motor_de_videojuego)

[7] Unity Technologies, Editor completo y ampliable, [En línea]. Avaliable: https://unity3d.com/es/unity

[8] Unity Technologies, Logo de Unity, [En línea]. Avaliable: https://unity.com/es

[9] Wikipedia, Docker (software), [En línea]. Avaliable: https://es.wikipedia.org/wiki/Docker_(software)

[10] Docker Inc, Docker, [En línea]. Avaliable: https://www.docker.com/

[11] Fiware, Plataforma Fiware, [En línea]. Avaliable: https://fiware-training.readthedocs.io/es_MX/latest/ecosistemaFIWARE/plataformaFIWARE/

[12] Fiware, Bienvenido a Orion Context Broker, [En línea]. Avaliable: https://fiware-orion.readthedocs.io/en/master/

[13] Fiware, Fiware-Cygnus, [En línea]. Avaliable: https://fiware-cygnus.readthedocs.io/en/latest/

[14] Oracle Corporation, MySQL, [En línea]. Avaliable: https://www.mysql.com/

[15] Wikipedia, C Sharp, [En línea]. Avaliable: https://es.wikipedia.org/wiki/C_Sharp

[16] Wikipedia, Python, [En línea]. Avaliable: https://es.wikipedia.org/wiki/Python

[17] Oracle, Java, [En línea]. Avaliable: https://www.java.com/es/download/faq/whatis_java.xml

[18] Wikipedia, Spring Framework, [En línea]. Avaliable: https://es.wikipedia.org/wiki/Spring_Framework

[19] Fiware, Desarrolla tu primera aplicación en Fiware, [En línea]. Avaliable: https://fiware-training.readthedocs.io/es_MX/latest/casodeestudio/descripcion/

[20] Docker, Hub Docker, [En línea]. Avaliable: https://hub.docker.com

[21] Amazon, Medidor pH Digital, [En línea]. Avaliable: https://www.amazon.es/gp/product/B07QKK1XB6/ref=ppx_od_dt_b_asin_title_s00?ie=UTF8&psc=1

[22] Microchip Technology Inc, MCP3004/3008, [En línea]. Avaliable: https://cdn-shop.adafruit.com/datasheets/MCP3008.pdf

[23] Adafruit Industries, Adafruit repository, [En línea]. Avaliable: https://github.com/adafruit

[24] Amazon Inc, Keyless Lock Wafu Cerradura eléctronica, [En línea]. Avaliable: https://www.amazon.es/electrónica-inteligente-móvil-Desbloqueo-smartphone-Cerrojo-antirrobo

[25] Fiware, Fiware NGSI-9 Open RESTful API Specification, [En línea]. Avaliable: https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/FI-WARE_NGSI-9_Open_RESTful_API_Specification

[26] Fiware, Update Action Types, [En línea]. Avaliable: https://fiware-orion.readthedocs.io/en/master/user/update_action_types/index.html

[27] Asociación Programo Ergo Sum, Control de GPIO con Python en Raspberry Pi, [En línea]. Avaliable: https://www.programoergosum.com/cursos-online/raspberry-pi/238-control-de-gpio-con-python-en-raspberry-pi/que-es-gpio

[28] Adafruit Industries, Analog Inputs for Raspberry Pi Using the MCP3008, [En línea]. Avaliable: https://cdn-learn.adafruit.com/downloads/pdf/reading-a-analog-in-and-controlling-audio-volume-with-the-raspberry-pi.pdf

[29] Unity Technologies, Descargar Unity, [En línea]. Avaliable: https://unity3d.com/es/get-unity/download

[30] Unity Technologies, Archivo de descarga de Unity, [En línea]. Avaliable: https://unity3d.com/es/get-unity/download/archive

[31] Oracle, Descargas del kit de desarrollo Java SE 8, [En línea]. Avaliable: https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

[32] Developers, NDK de Android, [En línea]. Avaliable: https://developer.android.com/ndk

[33] Unity Technologies, Unity 2019, [En línea]. Avaliable: https://unity3d.com/es/unity?_ga=2.142737653.1183569027.1567423094-204106923.1555518694

[34] OpenWebinars, Descarga e instalación de Spring Tools Suite sobre eclipse, [En línea]. Avaliable: https://openwebinars.net/blog/descarga-e-instalacion-de-spring-tools-suite-sobre-eclipse/

[35] The Apache Software Foundation, Bienvenido a Apache Maven, [En línea]. Avaliable: https://maven.apache.org/

[36] Pivotal Software.Inc, Spring Tools 4, [En línea]. Avaliable: https://spring.io/tools

[37] Escuela Superior de Ingenieria US, Guía Máquina Virtual,

[38] ZooPlus, Tetra AcuaSafe [En línea]. Avaliable: https://www.zooplus.es/shop/tienda_peces/limpieza_del_agua/purificador_agua

[39] Cloudi NextGen S.L. ,Tranajando con imágenes en Docker, [En línea]. Avaliable: https://clouding.io/kb/trabajando-con-imagenes-en-docker/

[40] Unity Technologies, Event Functions, [En línea]. Avaliable: https://docs.unity3d.com/es/current/Manual/EventFunctions.html

[41] Hibernate, Hibernate, [En línea]. Avaliable: http://hibernate.org/

[42] GitHub, Leito124 Trabajo, [En línea]. Avaliable: https://github.com/leito124/Spring-Trabajo