

## TECNOLOGÍAS APLICADAS EN SISTEMA CLIENTE-SERVIDOR PARA OFRECER AMBIENTES VIRTUALES CON DOCKER

Amaury Ortega<sup>1</sup>, Ramiro Verbel<sup>1</sup>, John Arrieta<sup>2</sup>

<sup>1</sup>Estudiante Ingeniería de Sistemas

<sup>2</sup>Profesor de Seminario de Actualización

---

### Resumen

En el presente trabajo se describe el entorno del sistema desarrollado dicho entorno se basa en una comunicación entre cliente-servidor en la cual el cliente solicita servicios y el servidor los proporciona estos servicios son Apache y MySQL, además se describen las tecnologías aplicadas en este, dichas tecnologías facilitan la comunicación del modelo cliente-servidor, gracias a la guía del docente se implementa el uso de Socket, Thread, JavaServer Pages junto a la comunicación mediante peticiones GET o POST, escritura y serialización de objetos a archivos JSON usados como mensajes de intercambio, WebServices que facilitan la construcción e implementación de aplicaciones distribuidas basadas en servicios Web HTTP y por último notificaciones Android usando Firebase Cloud Messaging de Google. Todas estas tecnologías de comunicación tienen sus puntos fuertes y débiles, además de relaciones y diferencias.

**Palabras claves:** servidor, cliente, comunicación, contenedor, servicios, MySQL, Apache, tecnología.

### Abstract

In the present work the environment of the developed system is described, the environment is based on a communication between the client and the server in which the client requests the services and the server provides them with these services. Apache and MySQL, in addition to the technologies applied in this, these technologies facilitate the communication of the client-server model, thanks to the teacher's guide the use of Socket, Thread, JavaServer Pages is implemented along with the communication through GET or POST requests, writing and serialization of objects to JSON files used as exchange messages, WebServices that facilitate the construction and implementation of distributed applications in HTTP Web services and, lastly, android notifications using Google's Firebase Cloud Messaging. All these communication technologies have their strengths and weaknesses, as well as relationships and differences.

**Keywords:** server, client, communication, container, services, MySQL, Apache, technology.

---

## 1. INTRODUCCIÓN

Nos enfocaremos en las tecnologías usadas para lograr la comunicación cliente-servidor con la finalidad de contrastar las diferentes tecnologías y determinar cuál es la más apropiada para prestar los servicios de contenedores virtuales.

## 2. OBJETIVOS

- Comprender la comunicación de red usando diferentes tecnologías con el fin de implementar un modelo cliente – servidor el cual proporcione instancias de servidores virtuales que contengan phpMyAdmin y MySQL accesibles mediante red por los clientes.

- Determinar la tecnología de comunicación eficiente para lograr el primer objetivo.

## 3. MARCO DEL PROYECTO

### 3.1. SERVIDOR

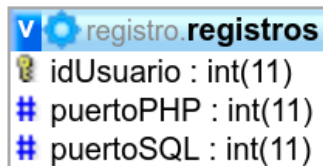
Es el encargado en procesar las peticiones de los clientes que son enviadas mediante tecnologías de comunicación, dichas petición son contestadas con la creación de un contenedor con los servicios de Apache y MySQL y sus respectivas formas de acceso, o el proceso inverso, la destrucción de un contenedor.

### 3.2. CLIENTE

Es el ente que solicita la creación o destrucción de contenedores virtuales que le proporcionan los servicios de apache y MySQL.

### 3.3. BASE DE DATOS

Se lleva un registro de los servidores virtuales asignados, en una base de datos MySQL conformada de la siguiente manera:



registro	
idUsuario	: int(11)
puertoPHP	: int(11)
puertoSQL	: int(11)

Ilustración 1. Tabla registros de la Base de Datos.

### 3.4. DOCKER

Docker es una herramienta open-source que permite realizar una ‘virtualización ligera’, con la cual se empaquetan entornos y aplicaciones que posteriormente son desplegables en cualquier sistema que disponga de esta tecnología.

Para ello Docker usa LXC (Linux Containers), es un sistema de virtualización que permite crear múltiples sistemas totalmente aislados entre sí, sobre la misma máquina.

La gran diferencia es que una máquina virtual necesita contener todo el sistema operativo mientras que un contenedor Docker aprovecha el sistema operativo sobre el cual se ejecuta compartiendo el kernel.

Se hace uso de la imagen wnameless/mysql-phpmyadmin:latest que viene con apache y phpmyadmin esta contiene configuraciones no funcionales para el proyecto, lo que conlleva a realizar cambios en dicha configuración, estos cambios se realizan bajo Ubuntu 12.04.

## 4. TECNOLOGÍAS USADAS PARA EL MODELO CLIENTE-SERVIDOR

### 4.1. SOCKET Y SERVERSOCKET

Se comunican a través de un canal TCP punto a punto, para esta comunicación el servidor y el cliente poseen un socket al extremo de su conexión permitiendo leer y escribir si estos se encuentran vinculados.

Para lograr forjar una mejor idea se define socket como el punto final de un enlace de comunicación bidireccional entre dos programas que se ejecutan en la red, donde un punto final es la combinación de un a dirección IP y un numero de puerto, cada conexión TCP se puede identificar de manera única por sus dos puntos finales de esta manera puede lograr múltiples conexiones entre el host y el servidor, estos se usan para representar la conexión cliente-servidor en donde el Socket se implementa del lado del cliente y el ServerSocket del lado del servidor.

Normalmente, un servidor se ejecuta en una computadora específica y tiene un socket que está vinculado a un número de puerto específico. El servidor simplemente espera a la escucha en el socket a que un cliente se conecte con una petición.

El cliente conoce el nombre de host de la máquina donde se ejecuta el servidor y el número de puerto donde el servidor está escuchando. Para realizar una solicitud de conexión, el cliente intenta reunirse con el servidor en la máquina y el puerto del servidor. El cliente también necesita identificarse con el servidor para que se vincule con un número de puerto local que usará durante esta conexión. Esto generalmente es asignado por el sistema.

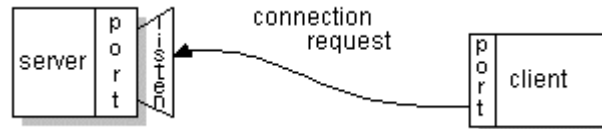


Ilustración 2. Solicitud de Conexión Cliente-Servidor

Si todo va bien, el servidor acepta la conexión. Pero antes, el servidor crea un nuevo socket en un puerto diferente. Es necesario crear un nuevo socket (y consecuentemente un número de puerto diferente) de forma que en el socket original continúe a la escucha de peticiones de nuevos clientes mientras se atiende las necesidades del cliente conectado. En el cliente, si se acepta la conexión, el socket se crea satisfactoriamente y se puede utilizar para comunicarse con el servidor.

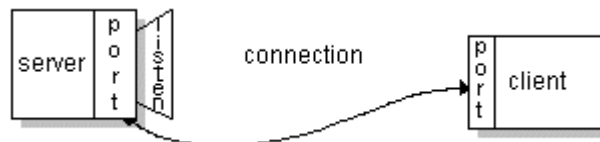
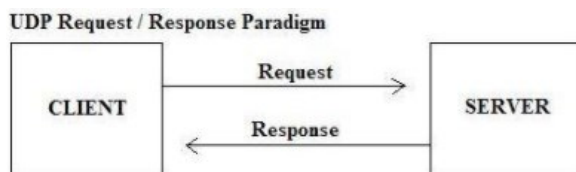


Ilustración 3. Conexión Cliente-Servidor

Para que el servidor pueda escuchar a varios clientes simultáneamente se hace uso de los Thread, cada vez que se establezca una nueva conexión se lanzara un nuevo hilo el cual se encarga de monitorizar la conexión entre el servidor y ese cliente, de esta manera el servidor podrá esperar nuevas conexiones.

## 4.2. DATAGRAMSOCKET Y DATAGRAM-PACKET

Se hace uso del protocolo UDP el cual proporciona un modo de comunicación de red mediante el cual las aplicaciones envían paquetes de datos, llamados datagramas, entre sí. Un datagrama es un mensaje independiente y autónomo enviado a través de la red cuya llegada, hora de llegada y contenido no están garantizados. Las clases DatagramPacket y DatagramSocket implementan comunicación de datagramas independiente del sistema usando UDP.



*Ilustración 3. Protocolo UDP*

En las comunicaciones basadas en datagramas, el paquete de datagramas contiene el número de puerto de su destino y UDP encamina el paquete a la aplicación apropiada, este destino es especificado usando un socket.

La clase DatagramPacket permite crear un datagrama y especificar el mensaje, la longitud del mensaje, la dirección Internet y el puerto local del socket de destino, además con ayuda de DatagramSocket permite recibir datagramas. En caso de que el mensaje sea muy grande el datagrama se trunca cuando llega, debido a que el protocolo IP permite longitudes de paquetes de máximo de  $2^{16}$  bytes, por lo que si necesitan mensajes mayores al máximo se deben fragmentar en pedazos del tamaño permitido.

La clase DatagramSocket da soporte a sockets para el envío y recepción de datagramas UDP. Mediante el uso de dos métodos send() y receive().

Estos métodos permiten transmitir datagramas entre un par de sockets. El argumento del send es una instancia de un DatagramPacket que contiene un mensaje y su destino. El argumento del receive es un objeto DatagramPacket vacío en el cual se pondrá el mensaje, su longitud y su origen.

Las comunicaciones mediante datagramas de UDP usan envíos no bloqueantes (non-blocking sends) y recepciones bloqueantes (blocking receives). Las operaciones de envío retornan cuando estas han dado el mensaje a los protocolos IP o UDP subyacentes, los cuales son responsables de transmitirlos a su destino. En la llegada, el mensaje es puesto en una cola por el socket que está asociado al puerto de destino. El mensaje puede ser recogido de la cola por una excepción o llamadas futuras de recepción (receive()) sobre ese socket. Los mensajes son descartados en el destino si ningún proceso tiene asociado un socket al puerto de destino. El método receptor (receive()) se bloquea hasta que se recibe un datagrama, a menos que se establezca un tiempo límite (timeout) sobre el socket. Si el proceso que invoca al método receive() tiene otra tarea que hacer mientras espera por el mensaje, debería planificarse en un flujo de ejecución (thread) separado.

## 4.3. JSP

JSP es un acrónimo de Java Server Pages, es una tecnología orientada a crear páginas web con programación en Java.

Con JSP podemos crear aplicaciones web que se ejecuten en variados servidores web, de múltiples plataformas, ya que Java es en esencia un lenguaje multiplataforma. Las páginas JSP están compuestas de código HTML/XML mezclado con etiquetas especiales para programar scripts de servidor en sintaxis Java. Por tanto, las JSP podremos escribirlas con nuestro editor HTML/XML habitual.

Al ser compilados son convertidos en servlet.

### 4.3.1. MOTOR JSP

El motor de las páginas JSP está basado en los servlets de Java (programas en Java destinados a ejecutarse en el servidor), en JSP creamos páginas generando archivos con extensión .jsp que incluyen, dentro de la estructura de etiquetas HTML, las sentencias Java a ejecutar en el servidor. Antes de que sean funcionales los archivos, el motor JSP lleva a cabo una fase de traducción de esa página en un servlet, implementado en un archivo class (Byte codes de Java). Esta fase de traducción se lleva a cabo habitualmente cuando se recibe la primera solicitud de la página .jsp, aunque existe la opción de precompilar en código para evitar ese tiempo de espera la primera vez que un cliente solicita la página.

### 4.3.2. SERVLET

Un servlet es una clase de lenguaje de programación Java utilizada para ampliar las capacidades de los servidores que alojan aplicaciones a las que se accede mediante un modelo de programación de solicitud y respuesta. Aunque los servlets pueden responder a cualquier tipo de solicitud, se usan comúnmente para extender las aplicaciones alojadas por los servidores web. Para tales aplicaciones, la tecnología Java Servlet define clases de servlet específicas de HTTP.

Son controlados por el contenedor (GlassFish, Tomcat) en el que es desplegado, al llegar peticiones a los servlet el contenedor comprueba si existe una instancia del servlet, si este no existe carga la clase del servlet, crea una instancia y es inicializada llamando al método `init()`, luego se invoca al método de servicio, pasándole objetos de tipo `request` y `response`, al finalizar para eliminar el servlet, el contenedor llama al método `destroy()` del servlet.

### 4.4. JSON

JSON es un acrónimo de JavaScript Object Notation, este es un formato de intercambio de información que está basado en estructuras de pares clave-valor. Es un formato mucho más ligero que XML y más indicado que éste en determinados escenarios.

Normalmente al intercambiar datos entre un navegador y un servidor, los datos solo pueden ser texto es aquí donde entra el poder de JSON ya que este es texto escrito con notación de objetos y podemos convertir cualquier objeto Java en JSON y enviarlo al servidor o viceversa, para esto tenemos GSON la cual es una librería open-source que nos permite serializar y deserializar cadenas de texto JSON, por tanto nos resulta útil para la comunicación cliente-servidor.

### 4.5. WEB SERVICE

Un servicio web (en inglés, Web service) es una pieza de software que utiliza un conjunto de protocolos y estándares que sirven para intercambiar datos entre programas. Distintas aplicaciones de software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma, pueden utilizar los servicios web para intercambiar datos en redes de ordenadores como Internet.

#### 4.5.1. WSDL

Son las siglas de Web Services Description Language, un formato XML que se utiliza para describir servicios Web. WSDL describe la interfaz pública a los servicios Web. Está basado en XML y describe la forma

de comunicación, es decir, los requisitos del protocolo y los formatos de los mensajes necesarios para interactuar con los servicios listados en su catálogo.

#### 4.5.2. SOAP

Las siglas de Simple Object Access Protocol, El protocolo estándar que se utiliza para enviar la información. Este define el formato del “envelope” que se intercambia entre cliente y servicio, así como las convenciones para representar invocaciones y respuestas. Estos mensajes son transmitidos en formato XML, montado sobre HTTP.

### 4.6. FIREBASE CLOUD MESSAGING

Es una solución de mensajería multiplataforma que permite enviar mensajes de forma segura y gratuita.

Con FCM, puedes notificar a una app cliente que un correo electrónico nuevo o que otros datos están disponibles para la sincronización. Puedes enviar mensajes de notificación para aumentar la captación y la retención de usuarios. Para los casos de uso de mensajería instantánea, un mensaje puede transferir una carga de hasta 4 KB a una app cliente.

#### 4.6.1. FUNCIONES

**Envía mensajes de notificación o mensajes de datos:** Envía mensajes de notificación que se muestran al usuario. También se puede enviar mensajes de datos y determinar completamente lo que ocurre en el código de tu aplicación. Consulta Tipos de mensajes.

**Orientación versátil de mensajes:** Distribuye mensajes a tu app cliente en cualquiera de las siguientes tres formas: a dispositivos individuales, a grupos de dispositivos o a dispositivos suscritos a temas.

**Envía mensajes desde apps cliente:** Envía mensajes de confirmación, de chat y de otros tipos desde los dispositivos a tu servidor a través del canal de conexión confiable de FCM que consume poca batería.

#### 4.6.2. COMO FUNCIONA

Una implementación de FCM incluye dos componentes principales para enviar y recibir datos:

Un entorno de confianza como Cloud Functions para Firebase o un servidor de apps para generar, orientar y enviar mensajes.

Una app cliente de iOS, Android o Web (JavaScript) que reciba mensajes.

Puedes enviar mensajes a través del SDK de Admin o las API de HTTP y XMPP. Para realizar pruebas o

enviar mensajes de participación o de marketing con orientación y análisis integrados potentes, también puedes usar el compositor de Notifications.

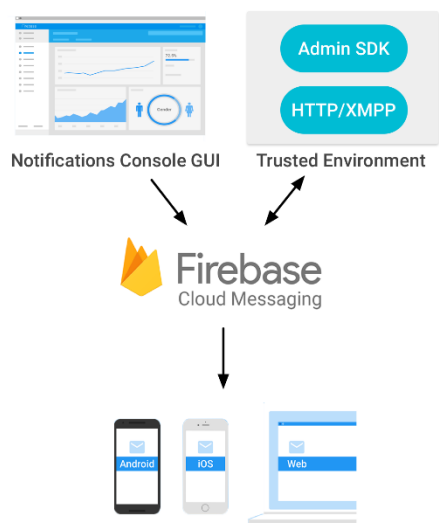


Ilustración 4. Funcionamiento de FCM

#### 4.6.3. RUTA DE IMPLEMENTACIÓN

**Configura el SDK de FCM:** Configura Firebase y FCM en tu app según las instrucciones de configuración para tu plataforma.

**Programa tu app cliente:** Agrega administración de mensajes, lógica de suscripción a temas y otras funciones opcionales a tu app cliente. Durante la etapa de programación, puedes enviar mensajes de prueba con facilidad desde el compositor de Notifications.

**Programa tu servidor de apps:** Decide si deseas usar el SDK de Admin o uno de los protocolos del servidor para crear la lógica de envío (lógica para autenticar, crear solicitudes de envío, administrar respuestas y otros). Luego, compila la lógica en tu entorno de confianza. Observa que, si deseas usar mensajería ascendente desde tus aplicaciones cliente, debes usar XMPP; además, Cloud Functions no admite la conexión continua que necesita XMPP.

### 5. RELACIONES, DIFERENCIAS Y PLUS DE LAS TECNOLOGÍAS

La implementación de socket se realizó mediante dos tipos diferentes los cuales tienen como principal diferencia el protocolo de comunicación usado, donde las clases Socket y ServerSocket utilizan TCP y DatagramSocket y DatagramPacket UDP, debido a esto se puede afirmar que Socket y ServerSocket presentan un

canal más seguro de comunicación que DatagramSocket y DatagramPacket.

DatagramSocket y DatagramPacket no garantiza que la información llegue a la hora correcta y completa, además no permite enviar mensajes grandes por lo que se deben fragmentar lo cual genera una complicación al momento de su programación y combinación con otras tecnologías y por ende hace más lenta la comunicación.

La tecnología de socket tiene el inconveniente que para poder escuchar a varios clientes simultáneamente necesita hacer uso de los Thread, cada vez que establezca una nueva conexión tendrá que lanzar un nuevo hilo, esto conlleva a un mayor uso de recursos del sistema.

JSP no posee una interfaz con muchas posibilidades, debido a la limitación de la interfaz web, la codificación de interfaces elaboradas ocupa demasiadas líneas de código y la mayoría debe hacerse mediante el uso de scripts, la carga de interfaces son relativamente más lentas que la de una aplicación de escritorio, la mayor parte de la lógica de la aplicación se ejecuta en el servidor, por lo que se corre el riesgo de sobrecargar el trabajo al mismo, la aplicación no está disponible si ocurre algún problema con el servidor o con la conexión de red.

JSP es un lenguaje totalmente escrito, posee una fuerte capacidad de manipulación de gráficos, permite la carga de APIs, es OpenSource y tenemos todas las ventajas del lenguaje java a nuestro alcance, lo cual permite una reusabilidad, robustez y multiplataforma.

Los Webservice aportan interoperabilidad entre aplicaciones de software independientemente de sus propiedades o de las plataformas sobre las que se instalen, además fomentan los estándares y protocolos basados en texto, que hacen más fácil acceder a su contenido y entender su funcionamiento. Al apoyarse en HTTP, los servicios Web pueden aprovecharse de los sistemas de seguridad firewall sin necesidad de cambiar las reglas de filtrado.

Los Web Service presentan un rendimiento bajo si se compara con otros modelos de computación distribuida, como RMI (Remote Method Invocation), CORBA (Common Object Request Broker Architecture), o DCOM (Distributed Component Object Model), además al apoyarse en HTTP, pueden esquivar medidas de seguridad basadas en firewall cuyas reglas tratan de bloquear la comunicación entre programas. Existe poca información de servicios web para algunos lenguajes de programación.

El formato de intercambio de información JSON tiene una gran facilidad de implementación gracias a

que solo se trata de texto plano estructurado en pares clave-valor, además es más ligero que otros formatos como XML y resulta de gran ayuda en determinados escenarios.

Gracias a FCM se le dice adiós a los WebService, ya que solo se necesita tener la lógica de la App móvil y un poco de conocimiento en base de datos. Debido a la facilidad de implementar la librería y utilizar los ejemplos de la documentación de FireBase.

Además es multiplataforma, de esta manera se puede utilizar la misma lógica del código para cada lenguaje de programación, debido a esto cuenta con altas capacidades de integración, sin mencionar el alto rendimiento que posee el cual resulta adecuado para aplicaciones en tiempo real.

Permite implementar notificaciones tipo Push de una manera intuitiva, logrando hacer uso de esta en pocas líneas de código.

Lo malo de esto es que FireBase tiene un límite de conexiones simultáneas en su plan gratuito un máximo de 100, por tener base de datos no complejas es difícil poder ejecutar consultas estructuradas en los datos, las herramientas se manejan en diferentes arquitecturas y enfoques, faltas de funciones optimizadas de búsqueda y consulta.

## 6. CONCLUSIONES

A partir del desarrollo del proyecto se identificaron las ventajas y desventajas de las diferentes tecnologías utilizadas con el fin de implementar el modelo cliente-servidor logrando así proporcionar instancias de servidores virtuales con determinadas herramientas, luego de sumergirse en todas estas tecnologías decidimos que la que mayor adaptabilidad tenía a nuestro proyecto era la combinación de JSP junto al formato Json y el uso de FCM con el fin de notificar en tiempo real.

## 7. ANEXOS

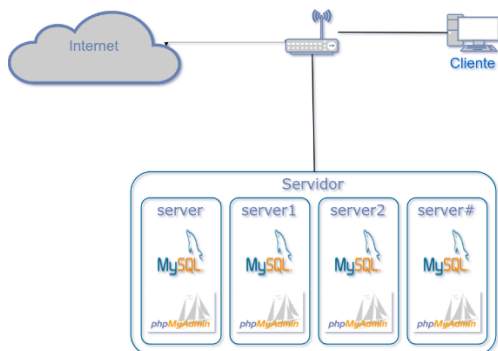


Ilustración 5. Arquitectura

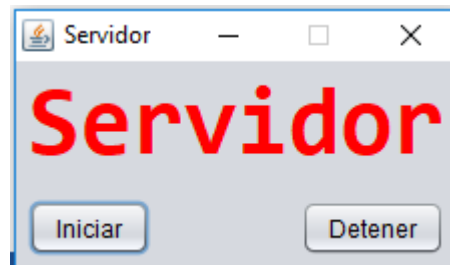


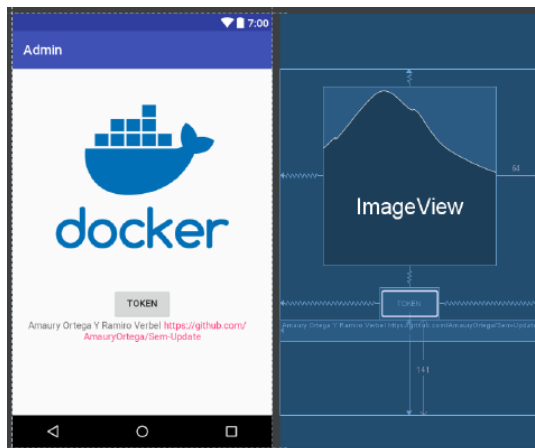
Ilustración 6. App Servidor



Ilustración 7. App Cliente



Ilustración 8. App Cliente Iniciada





## 8. REFERENCIAS

- [1] ORACLE. (s.f.). Class Socket. Obtenido de Java™ Platform: <https://docs.oracle.com/javase/7/docs/api/java/net/Socket.html>
- [2] ORACLE. (s.f.). Lesson: All About Sockets. Obtenido de ORACLE Java Documentation: <https://docs.oracle.com/javase/tutorial/networking/sockets/index.html>
- [3] ORACLE. (s.f.). Class DatagramPacket. Obtenido de Java™ Platform: <https://docs.oracle.com/javase/7/docs/api/java/net/DatagramPacket.html>
- [4] ORACLE. (s.f.). Class DatagramSocket. Obtenido de Java™ Platform: <https://docs.oracle.com/javase/7/docs/api/java/net/DatagramSocket.html>
- [5] ORACLE. (s.f.). Class HttpServlet. Obtenido de Java Platform, Enterprise Edition (Java EE) 7: <http://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpServlet.html>
- [6] Foundation, T. A. (s.f.). Apache HttpCore 4.4.7 API. Obtenido de Package org.apache.http: <https://hc.apache.org/httpcomponents-core-ga/httpcore/apidocs/org/apache/http/>
- [7] Foundation, T. A. (04 de Septiembre de 2017). Http Components. Obtenido de Inicio rápido de HttpClient: <https://hc.apache.org/httpcomponents-client-ga/quickstart.html>
- [8] León Hernández, C. (s.f.). Las clases Java DatagramPacket y DatagramSocket. Obtenido de Universidad de La Laguna: <http://nereida.deiocl.ull.es/~cleon/doctorado/doc06/doc06/html/node8.html>
- [9] Telematica, D. d. (s.f.). Trabajando con Sockets UDP. Obtenido de Universidad Carlos III de Madrid: <http://www.it.uc3m.es/celeste/documentacion/cr/2003/PracticaSocketsUDP/>
- [10] Mestras, J. P. (s.f.). Java EE – Servlets. Obtenido de Dep. Ingeniería del Software e Inteligencia Artificial. Universidad Complutense Madrid: <https://www.fdi.ucm.es/profesor/jpavon/web/43-servlets.pdf>
- [11] Google. (2017). Repositorio gson. Obtenido de <https://github.com/google/gson>
- [12] Google. (s.f.). Firebase Documentation FCM. Obtenido de Google Developers: <https://firebase.google.com/docs/cloud-messaging>
- [13] Google. (s.f.). Firebase Documentation Getting Started. Obtenido de Google Developers: [https://firebase.google.com/docs/android/setup?authuser=0#manually\\_add\\_firebase](https://firebase.google.com/docs/android/setup?authuser=0#manually_add_firebase)
- [14] Rodríguez, M. A. (17 de Septiembre de 2012). Jugando con JSON en Java y la librería Gson. Obtenido de <https://www.adictosaltrabajo.com/tutoriales/gson-java-json/>
- [15] Sanchez, J. (2 de Marzo de 2012). INGENIERIA DE SISTEMAS Y ELECTRONICA. Obtenido de Creacion de Un Webservice en Java: <https://ingsistele.wordpress.com/2012/03/02/creacion-de-un-webservice-en-java/>
- [16] Ortega Camargo, A. R. (2017). Repositorio Learning-Docker. Obtenido de GitHub.com: <https://github.com/AmauryOrtega/Learning-Docker>
- [17] Ortega Camargo, A. R. (s.f.). Docker Hub. Obtenido de [xxdrackler/xx/test-PUBLIC-REPOSITORY](https://github.com/xxdrackler/xx/test-PUBLIC-REPOSITORY): <https://hub.docker.com/r/xxdrackler/xx/test/>

- [18] Ortega Camargo, A. R., & Verbel de la Rosa, R. J. (2017). Repositorio Sem-Update. Obtenido de Github: <https://github.com/AmauryOrtega/Sem-Update>