UNIVERSIDAD POLITÉCNICA DE SINALOA
PROGRAMA ACADÉMICO DE
INGENIERÍA EN INFORMÁTICA

Ingeniería en Informática

Uso de la tecnología Java en el Internet Of Things

Autor

Saúl Alvarado Quiñonez

Asesora

MC. Iliana Amabely Silva Hernández

Mazatlán, Sinaloa 3 de diciembre de 2017

UNIVERSIDAD POLITÉCNICA DE SINALOA

SINALOA
SECRETARÍA DE EDUCACIÓN PÚBLICA Y CULTURA

C. ALVARADO QUIÑONEZ SAUL **Folio 2014030453-2017-033**
P r e s e n t e .-

Por medio de la presente me permito comunicarle que **es de aceptarse el tema de tesina,** el cuál se ha solicitado bajo el título:

" Uso de Java en Internet of Things "

mismo que usted desarrollará con objeto de dar lugar a los trámites conducentes para la acreditación de la asignatura de Estadías Profesionales de la Unidad Académica de:

Ingeniería en Informática

Así mismo, le comunico que para el desarrollo de la citada tesina le ha sido asignado como Director Asesor de la misma a la: **M.C. Iliana Amabely Silva Hernández** y como asesores a la **M.C. Gloria Irene Téllez Rodríguez** y al **M.C. Luis Armando Aguayo Medina**

Sin otro particular por el momento, aprovecho la ocasión para enviarle un cordial saludo.

Atentamente

Dr. Rodolfo Ostos Robles
Director del Programa Académico de Ingeniería en Informática
Universidad Politécnica de Sinaloa

C.c.p. Interesado

SEP
SECRETARÍA DE EDUCACIÓN PÚBLICA

UTyP
COORDINACIÓN GENERAL DE UNIVERSIDADES TECNOLÓGICAS Y POLITÉCNICAS

UNIVERSIDAD POLITÉCNICA DE SINALOA

SINALOA
SECRETARÍA DE EDUCACIÓN PÚBLICA Y CULTURA

C. ALVARADO QUIÑONEZ SAUL **Folio 2014030453-2017-033**
P r e s e n t e.-

Por este conducto le envío un cordial saludo y aprovecho la ocasión para notificarle que el jurado que le fue asignado para evaluar la tesina desarrollada en las estadías profesionales denominada ***" Uso de Java en Internet of Things "*** y que después de ser revisada en reunión de sinodales, ante la Dirección de la Unidad Académica de Ingeniería en Informática, integrada por:

PRESIDENTE DEL JURADO: M.C. Iliana Amabely Silva Hernández Iliana Amabely Silva Hernandez

SINODAL: M.C. Gloria Irene Téllez Rodríguez Gloria I. Tellez R.

SINODAL: M.C. Luis Armando Aguayo Medina

Ha decidido autorizar y aceptar la digitalización de la misma por el participante, conforme a la normatividad vigente y cumpliendo con los requisitos para tal caso.

Agradeciendo la atención a la presente, le reitero a Usted mi atenta consideración y respeto.

Atentamente

Dr. Rodolfo Ostos Robles
Director del Programa Académico de Ingeniería en Informática
Universidad Politécnica de Sinaloa

DIRECCIÓN INFORMATICA

SEP

UTP

tr3sco
soluciones móviles

www.tr3sco.com

CDMX a 4 de septiembre de 2017

M.C. MARÍA GUADALUPE GARCÍA RAMÍREZ
DIRECTORA DE VINCULACIÓN, DIFUSIÓN Y EXT. UNIVERSITARIA.
UNIVERSIDAD POLITÉCNICA DE SINALOA.

PRESENTE

Por este medio, hago de su conocimiento que el alumno(a) el **C. SAÚL ALVARADO QUIÑONEZ** con número de matrícula **2014030453**, de la carrera de Ingeniería en Informática, ha sido aceptado para realizar su estadía práctica, en esta empresa, durante el período que comprende del 4 de septiembre al 20 de diciembre, para cubrir un total de 600 horas.

Dicho alumno realizará actividades dentro del área de back-end, bajo la supervisión del C. ALEJANDRO QUINTANA OSUNA CEO.

Sin otro particular, le envío un cordial saludo.

Atte.

Alejandro Quintana O.

Alejandro Quintana Osuna

tr3sco
soluciones móviles

tr3sco
soluciones móviles

www.tr3sco.com

CDMX, 8 de diciembre de 2017

M.C. MARÍA GUADALUPE GARCÍA RAMÍREZ
DIRECTORA DE VINCULACIÓN, DIFUSIÓN Y EXT. UNIVERSITARIA.
UNIVERSIDAD POLITÉCNICA DE SINALOA.

PRESENTE

Por este medio, hago de su conocimiento que el alumno(a) el C. Saúl Alvarado Quiñonez con número de matrícula 2014030453, de la carrera de Ingeniería en Informática, ha cumplido con 600 horas correspondientes a estadía final, en esta empresa/institución, durante el período que comprende del 4 de septiembre 2017 al 8 de diciembre 2017.

Dicho alumno realizó actividades dentro del área/departamento de Desarrollo, bajo la supervisión del C. Alejandro Quintana Osuna CEO de TR3SCO.

Sin otro particular, le envío un cordial saludo.

Atte.

Alejandro Quintana O.

Alejandro Quintana Osuna

tr3sco
soluciones móviles

Dedicatoria

Dedico este documento a todas aquellas personas amantes de la tecnología, aquellas personas que creen fielmente en el progreso y que están dispuestas a cambiar el mundo y mejorar nuestro estilo y calidad de vida como civilización.

Agradecimientos

Agradezco a mi hogar de formación profesional, la Universidad Politécnica de Sinaloa por cultivar en mí el espíritu del trabajo duro, curiosidad, innovación, creatividad y desarrollo humano; así como también el amor por mi profesión.

Muchas gracias.

Índice temático

Contenido

Índice de figuras

Índice de fotografías

Índice de tablas

Resumen

La presente tesina pretende dar a conocer las ventajas y desventajas de la implementación de la tecnología Java en el IOT (Internet Of Things) con el fin de lograr un IOT más integrado, accesible, amplio, seguro y responsivo para millones de dispositivos finales. Java es ejecutado por billones de dispositivos lo cual lo transformaría en un ecosistema de ejecución amigable para los dispositivos finales ejecutando Java. Desde hace ya 20 años Java es uno de los lenguajes de programación líderes en el mercado, y es utilizado por grandes empresas por la gran estabilidad, rapidez y seguridad que ofrece. Durante la presente tesina se verá el uso que ha tenido Java para el Internet Of Things en los últimos años incluso por encima de otros lenguajes, también se verán sus diferencias y preferencias entre los desarrolladores en cuanto a que lenguaje utilizar para el Internet Of Things en casos muy específicos.

Palabras clave: Java, Internet Of Things, dispositivo, ecosistema.

Abstract

This thesis aims to announce the advantages and disadvantages about the implementation of the Java technology in the IOT (Internet of Things) in order to achieve an integrated, accessible, wide, secure and responsive IOT for millions of devices. Java runs in billions of devices, which would transform it in a friendly runtime environment for final devices that runs Java. Since 20 years ago, Java is one of the programming languages leaders in the market and used by the enterprises due to its stability, speed y security. During this thesis, we will aboard the use that Java has had in the last years for the Internet of Things even above other programming languages; also, we will see the differences and preferences in the developer community about which language to use in the different cases for the Internet of Things.

Keywords: Java, Internet of Things, device, environment.

Capítulo I

Introducción

El IOT (Internet Of Things) ha dado mucho de qué hablar en la segunda década del año 2000, sin embargo, es un tema que se introdujo a finales de los años 90 y que desde aquél entonces ha ganado popularidad y más recientemente, gracias a la introducción de los Smarthphones, que han hecho posible tener un mundo totalmente conectado, no obstante, el hecho de exponer un dispositivo al IOT puede resultar en un arma de doble filo si no se tiene un riguroso control de seguridad.

Al estar conectados a una red tan grande como internet podemos hacer maravillas con tanta información, desde consultar un tema desconocido hasta conocer nuestro historial de visitas, todo ello almacenado en una computadora corriendo el riesgo de que otra persona con malas intenciones logre penetrar la seguridad de la misma y extraer dicha información. Ahora bien, imaginemos una poderosa tecnología que exista en billones de dispositivos alrededor del mundo con una considerable cantidad de años de soporte y mantenimiento, una tecnología que promete un flujo de información alto, un ecosistema mundialmente coordinado con dispositivos finales utilizando la misma tecnología, respuesta en tiempo real y una confiable seguridad de punto a punto, el nombre de esta tecnología es Java y debido a su gran popularidad y su uso en billones de dispositivos la convierte en un candidato perfecto a considerar para la implementación con el IOT.

Antecedentes

Se comenzó a considerar IOT cuando empezaron a haber más “cosas” conectadas a Internet que personas. Ya en 1990 John Romkey creó el primer objeto conectado a Internet, una tostadora que podía encenderse o apagarse remotamente, sin embargo, la comunicación que Internet ofrecía en aquél entonces eran principalmente cableada sumado a que los costes de hardware eran muy elevados hicieron pasar desapercibidas las ideas de tener objetos conectados durante varios años. No fue hasta la llegada de los smartphones y la popularización de la conectividad inalámbrica que las ideas de tener objetos conectados resurgieron, lo cual poco a poco originaría un entorno conectado en el cual no sólo participarían dispositivos móviles, sino también cualquier objeto con electrónica de control que nos permitiera conocer donde se encuentra, interactuar con ellos, obtener información de su estado, etc. El IOT continúa en sus inicios y se están utilizando las tecnologías que existen en su momento, pero al mismo tiempo se está llevando a cabo con nuevas tecnologías, redes, protocolos y dispositivos, entre los cuales está la tecnología Java.

Planteamiento del problema

Actualmente el IOT es una tendencia que continúa siendo desarrollada, por lo cual aún presenta fallas, muchas de ellas catalogadas como vulnerabilidades de seguridad potencialmente peligrosas, las cuales exponen información personal y que eventualmente se volverán más complejos de resolver debido al masivo crecimiento del IOT; se propone utilizar la tecnología Java no sólo con el fin de ofrecer un ecosistema agradable entre dispositivos, sino más bien por la alta seguridad punto a punto que ofrece en su ambiente de ejecución.

Hipótesis

En este estudio se espera demostrar las capacidades y alta fiabilidad de la tecnología Java, intentando resolver el principal problema del IOT, la seguridad; así como también mejorar otras características de la misma.

Objetivos

- Implementar la tecnología Java en una solución IOT.
- Comprobar la superioridad de Java ante otros lenguajes de programación como alternativa para el IOT.

Importancia y/o justificación del estudio

Este estudio es llevado a cabo con el fin de dar a conocer la fiabilidad de una tecnología que lleva años en el mercado, la cual ha sido usada por billones de dispositivos y que ha demostrado que puede persistir en diferentes ambientes de ejecución, una tecnología que puede ser utilizada para un IOT más rápido, confiable y seguro.

Limitaciones del estudio

Informar acerca de los beneficios de utilizar la tecnología Java como una alternativa para el IOT e implementar una aplicación aplicando el IOT con java.

Definición de términos

- **IOT:** Internet Of Things.
- **IOE:** Internet Of Everything.
- **Java:** Lenguaje de programación creado por Sun Microsystems el cual fue comprado por Oracle.
- **Java SE:** Java Standard Edition.
- **Java ME:** Java Micro Edition.
- **Ambiente de ejecución:** Entorno de ejecución de Java (JRE, Java Runtime Environment) es el ambiente de software en el cual se ejecutan las tareas de Java.
- **Web Service:** Servicio que es ejecutado en la nube, es decir, un código que está en la nube y retorna una respuesta.
- **API:** Application Programming Interface, es un conjunto de servicios en la nube.
- **AWS:** Amazón Web Services.
- **End-point:** Un punto final que puede o no estar localizado en un lugar remoto.

- **Backend:** Es el código de la aplicación que no puede ser visto por el cliente.
- **Cliente:** Una computadora que realiza una petición a un servidor, por lo regular suelen ser los usuarios de una aplicación.
- **Servidor:** Es aquella computadora que recibe las peticiones del cliente y retorna lo que haya solicitado.
- **IP:** Internet Protocol, es un número único e irrepetible que es asignado a una computadora dentro de una red.
- **TCP:** Transmission Control Protocol, se utiliza para crear conexiones entre dispositivos.
- **IDE:** Integrated Development Environment, es una aplicación que facilita el desarrollo de Software.
- **Gateway:** Es la salida que una red tiene a Internet.
- **Bit:** Mínima unidad de almacenamiento de una computadora (1, 0).

Capítulo II

Estado del arte

El primer invento conocido del IOT fue una tostadora que trabajaba sobre el protocolo TCP/IP y SNMPMIB (Simple Networking Management Protocol Management Information Base) la tostadora podía ser encendida o apagada remotamente, no obstante, aún se necesitaba la ayuda de una persona para colocar el pan en la misma. En 1991 se agregó una pequeña grúa robótica al sistema (controlada desde Internet) que permitía tomar una rebanada de pan y meterla a la tostadora automatizado el proceso.

En los últimos 5 años el IOT ha adquirido una especial importancia en los dispositivos y las grandes empresas como Google, Cisco, Intel, IBM y BOSCH, estás empresas apuestan por ello desarrollando tecnologías para el IOT. A continuación, se muestran los resultados de las encuestas IOT developer de los años 2015, 2016 y 2017.

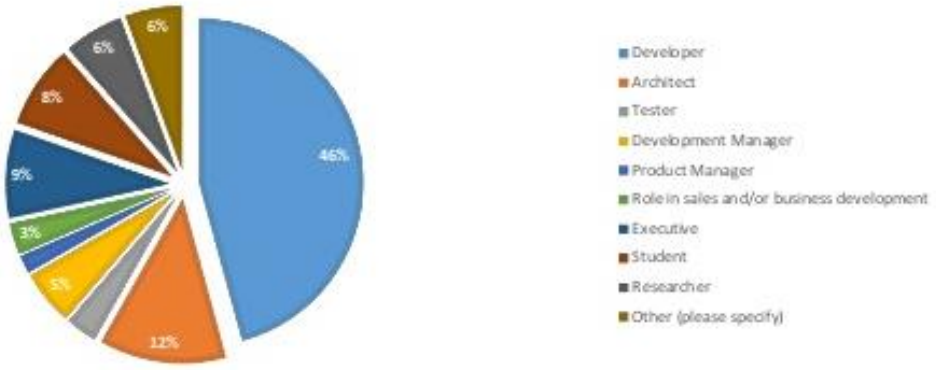
IOT Developer 2015
Resultados clave

- **Los desarrolladores están muy comprometidos con el IOT**, el 33% de los desarrolladores encuestados están creando soluciones para el IOT, mientras que alrededor del 50% están aprendiendo y descubriendo nuevas tendencias.
- **TOP 3 influenciadores para las tecnologías del IOT**, Vendedores de hardware, manufactureras de semiconductores y los proveedores de almacenamiento en la nube.
- **Líderes del IOT**, Google, Cisco, Intel, IBM y BOSCH.
- **Los consorcios más importantes del IOT**, Eclipse IOT, IEEE e IETF.
- **TOP 3 preocupaciones del IOT**, Seguridad, interoperabilidad y la integración del Hardware.
- **La presencia del Open Source en el IOT**, el 81% de las soluciones con IOT son hechas con tecnologías Open Source.
- **Tecnologías clave para el IOT**, HTTP y MQTT para el envío de información y Linux como sistema operativo.

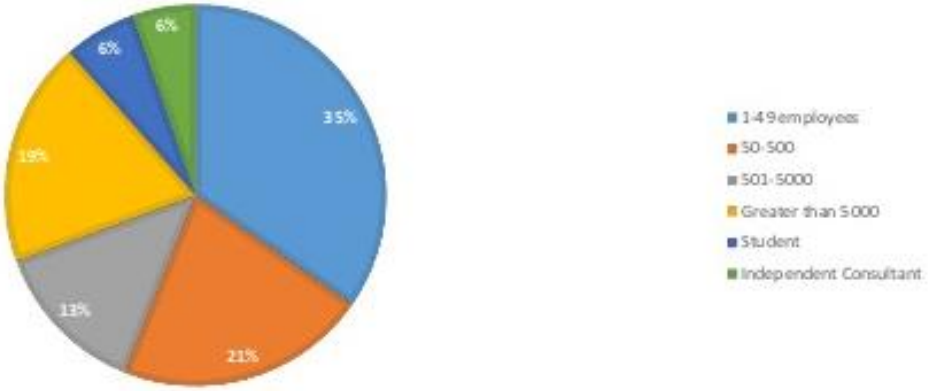
- **TOP 3 lenguajes para el IOT**, Java, C y Javascript

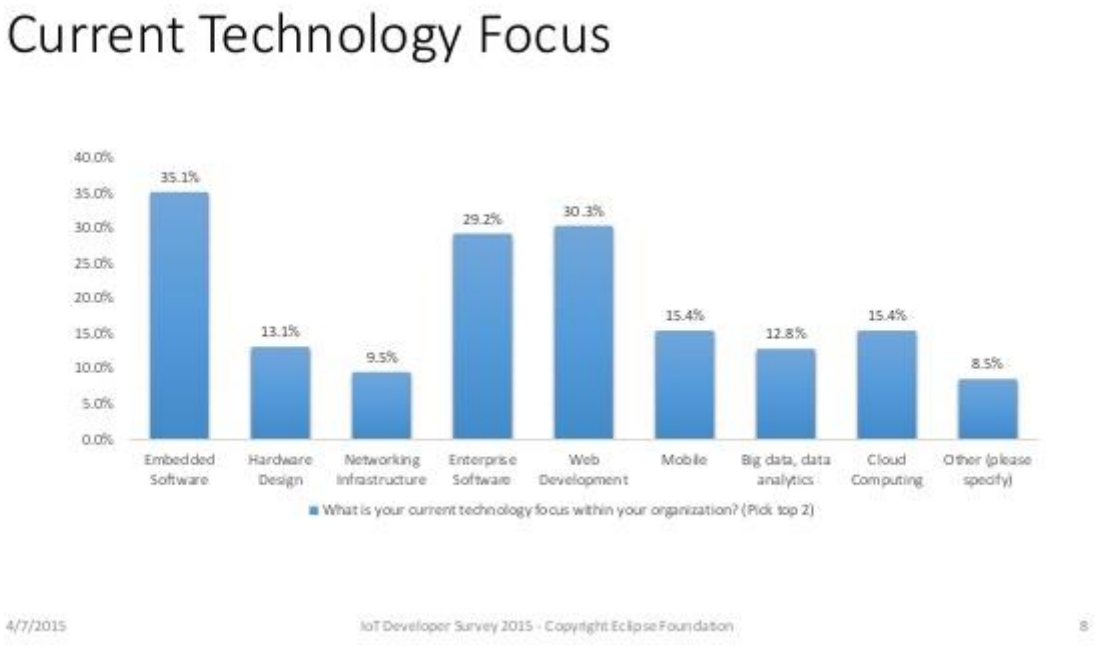
¿Qué dicen los profesionistas?

La encuesta fue aplicada a personas de diferentes profesiones, de entre las cuales las más destacadas fueron los desarrolladores con un 46%, arquitectos con un 12% y los ejecutivos con un 9%.



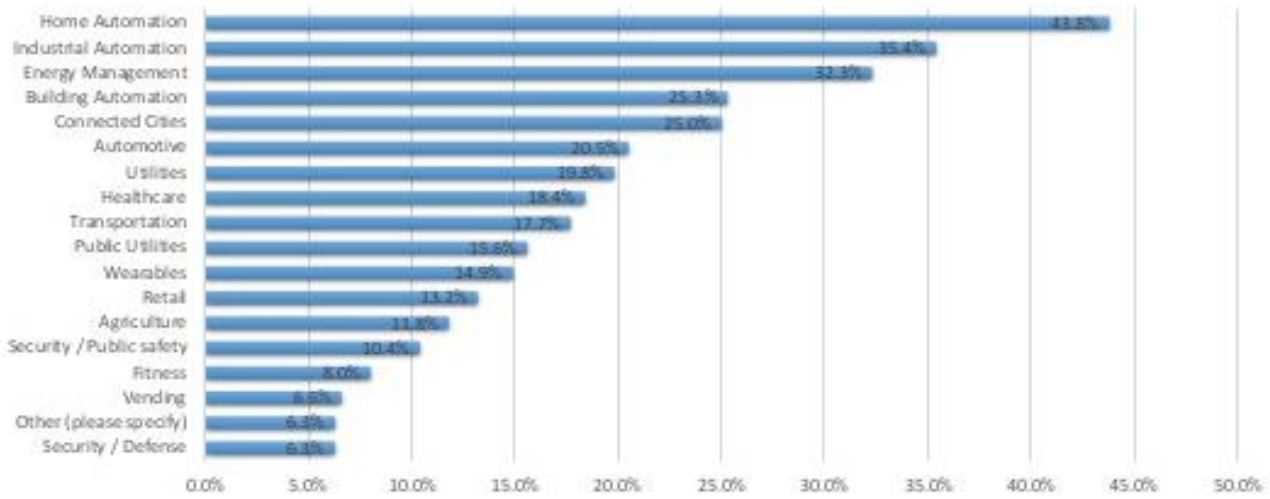
En empresas con una cantidad de recursos humanos de 1 a 49 y de 50 a 500. Los enfoques de las tecnologías en las cuales se aplica el IOT son en 35.1% al Software embebido, 30.3% al Desarrollo Web y 29.2% al Software empresarial. Se menciona que el 45% de las empresas donde estos profesionistas laboran ya desarrollan e implementan soluciones con IOT, mientras que el resto tienen planeado desarrollar soluciones IOT en los próximos 6 o 18 meses o incluso algunas no tienen planes de desarrollar soluciones IOT.





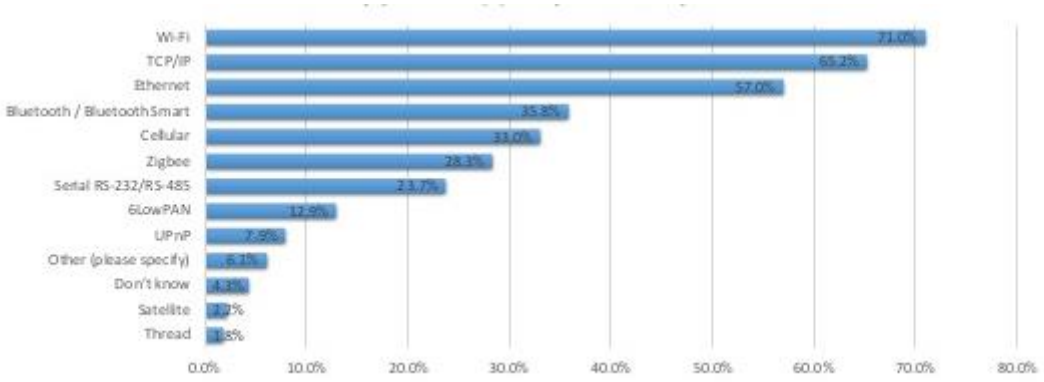
A cada persona encuestada se le hizo la pregunta de “¿Cuál es tu experiencia en soluciones IOT?” a la cual el 33% respondió que desarrollan soluciones IOT para la compañía en la que trabajan, el 22% respondió que aprenden de las tecnologías IOT en su tiempo libre y el 15% respondió que desarrollan soluciones IOT en su tiempo libre.

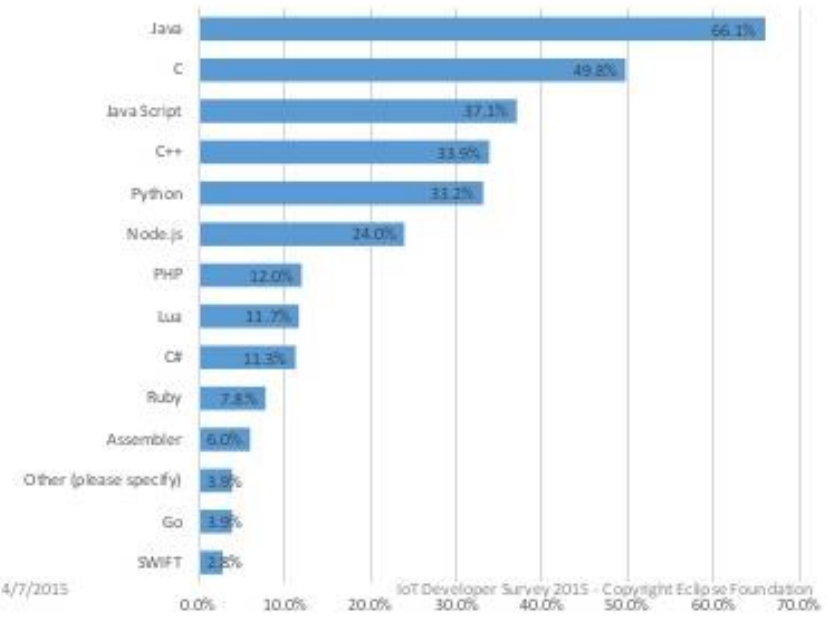
A continuación, se muestran las áreas de soluciones más acudidas:



Tecnologías del IOT

En las tecnologías usadas se encuentra en primer lugar como lenguaje de programación Java, le sigue C y Javascript, sin embargo, cabe mencionar que para el desarrollo en software embebido el lenguaje C# es el que lidera con un 64% j Java le sigue por debajo de este con un 58%. Los protocolos de mensaje más utilizados con en las soluciones IOT son HTTP, MQTT y CoAP, en los protocolos de conectividad se encuentran como lideres el Wi-Fi, TCP/IP, Ethernet y Bluetooth, finalmente, como sistema operativo líder en desarrollo de soluciones se encuentra Linux, muy por encima del resto con u 78.2% de uso.





For Embedded Software developers C is #1 language (64%) and then Java (58%)

4/7/2015 IoT Developer Survey 2015 - Copyright Eclipse Foundation 17

IOT Developer 2016
Resultados clave

Las compañías ya están implementando soluciones IOT

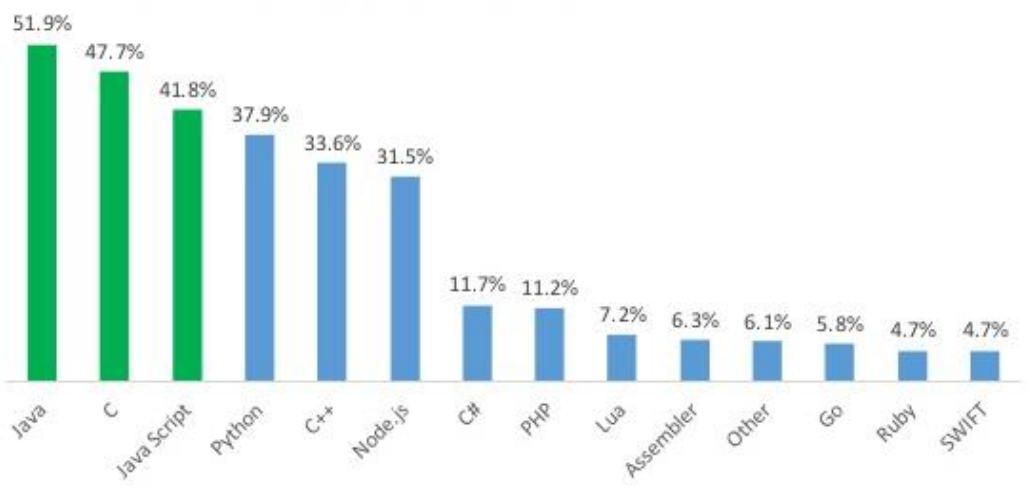
El 46% de los encuestados responde que la compañía para la que trabajan ya se encuentra desarrollando e implementando soluciones IOT, otro 29% dice que planean hacerlo en un periodo de 6 meses. Esta podría ser una clara indicación de que la industria está madurando rápidamente.

La seguridad continúa siendo una preocupación

No es de esperarse la seguridad continúe siendo una gran preocupación en el IOT, sin embargo, la segunda gran preocupación de los desarrolladores es la interoperabilidad, aunque puede ser que los grandes problemas de la interoperabilidad estén a punto de ser solucionados gracias a proyectos como **Eclipse Hono, Eclipse Smarthome y Eclipse Akura**. No obstante, parece ser que las empresas aún deben enfocarse de lleno en atender los problemas de la seguridad, debido a que es un problema que necesitar ser resuelto. Otro detalle a destacar es que para aquellas empresas que ya han implementado soluciones IOT el rendimiento es aquello que se está transformando en le tercera preocupación, aunque no está claro cuáles son exactamente los problemas de rendimiento, pero es algo que es más que seguro que será investigado.

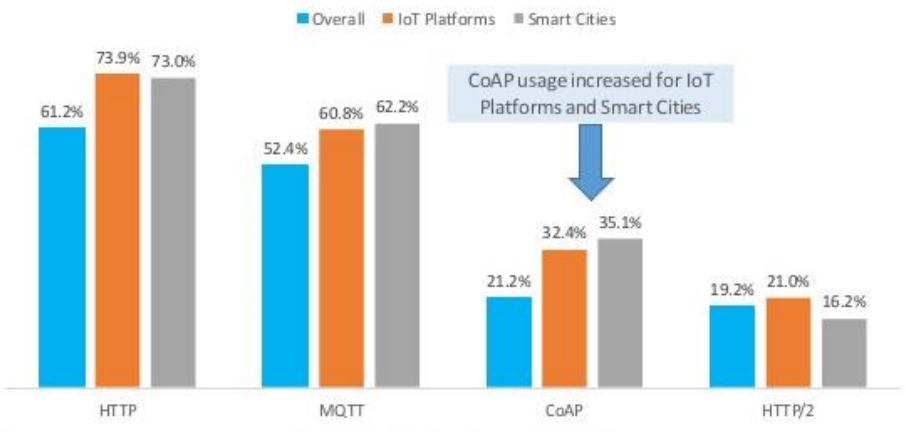
Top lenguajes de programación para el IOT

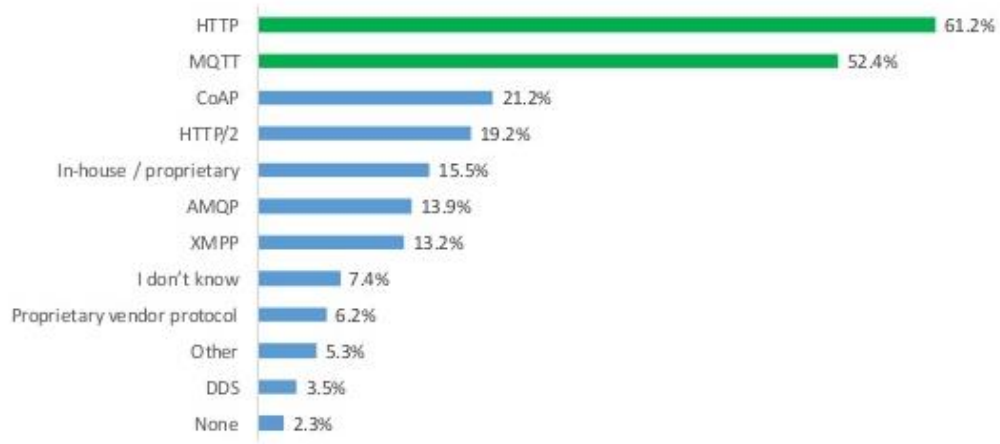
Java, C, Javascript y Python, no es una sorpresa ver a estos lenguajes como los más utilizados, aunque muchas personas cuestionan el uso de Java para el IOT la verdad es que hay una gran comunidad que ha hecho proyectos utilizando Java como lenguaje de programación, por lo tanto, hay una cierta inconformidad con Java para aquellos que no suelen utilizarlo mucho. Sin embargo, si quitamos a estas personas que no lo reconocen como buena opción, Java seguiría en el Top 3 de lenguajes junto a C y Python.



MQTT y HTTP continúan siendo los protocolos dominantes para el envío de mensajes

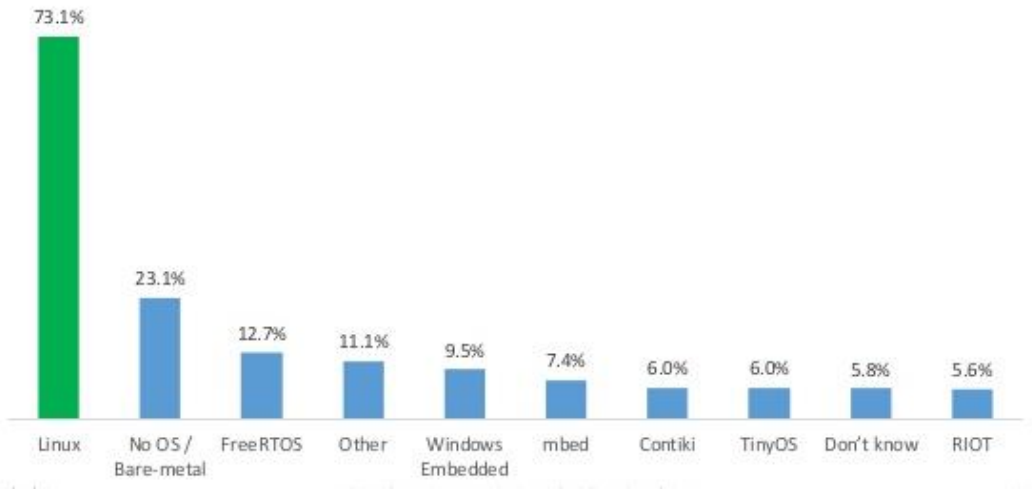
Sin duda alguna MQTT se ha convertido en un protocolo utilizado ampliamente para el IOT, junto a HTTP que va por encima de este. Otro protocolo que también suele ser usado en la comunidad de desarrolladores es CoAP que no ha recibido mucho soporte últimamente, pero parece tener soporte sólo en algunos sectores de la industria. Por ejemplo, CoAP es más utilizado para los sectores del desarrollo de plataformas IOT o en las ciudades inteligentes. Por otro lado, el éxito del protocolo MQTT se debe a una estrategia de IBM que hoy en día gracias a esta estrategia MQTT recibe soporte tanto como de IBM, Amazon AWS IOT, Microsoft Azure IOT, entre otras plataformas IOT en el mercado, otro punto a destacar es que la nueva placa de Arduino también utiliza MQTT para comunicarse con la nube.





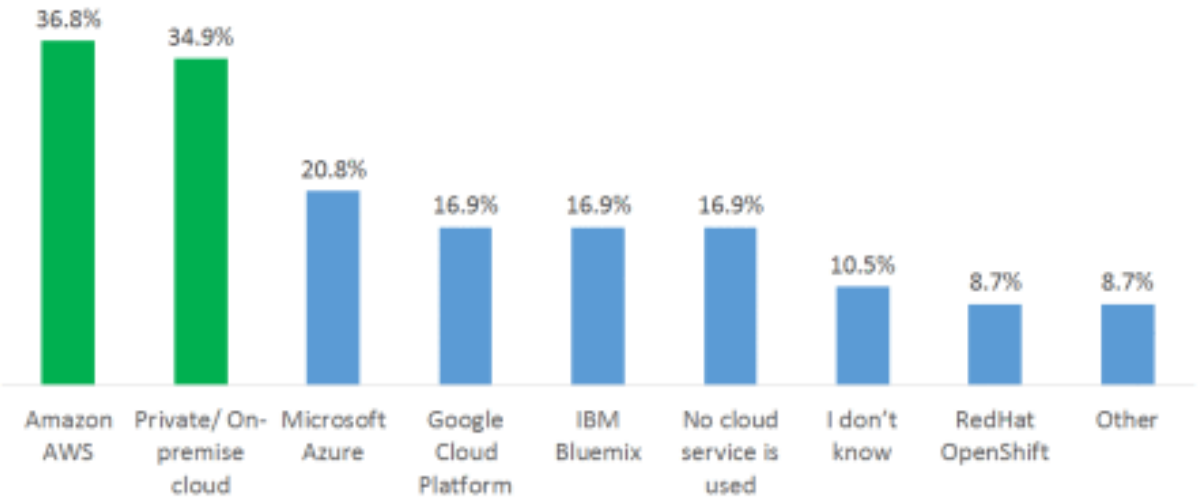
Linux es el sistema operativo dominante

Más del 70% de los encuestados respondieron que usan Linux como sistema operativo para las soluciones IOT. La siguiente sección más popular después de Linux es un sistema operativo llamado No OS/Bare metal. En los últimos años, un gran número de sistemas operativos IOT han sido introducidos al mercado, aunque estos aún no levantan gran demanda para los desarrolladores sería interesante como estos sistemas operativos crecen en cuanto a uso en comparación con Linux.



Amazon lidera en servicios en la nube para IOT

Amazon se ha vuelto el número como proveedor de servicios en la nube, sin embargo, los servicios privados parecen no estar muy lejos de Amazon, creo que esto se debe a que los servicios IOT en la nube todavía continúan en un proceso de "definición", sorpresivamente podemos ver a Microsoft Azure en la tercera posición, esto refleja que Microsoft también está enfocándose en el IOT.

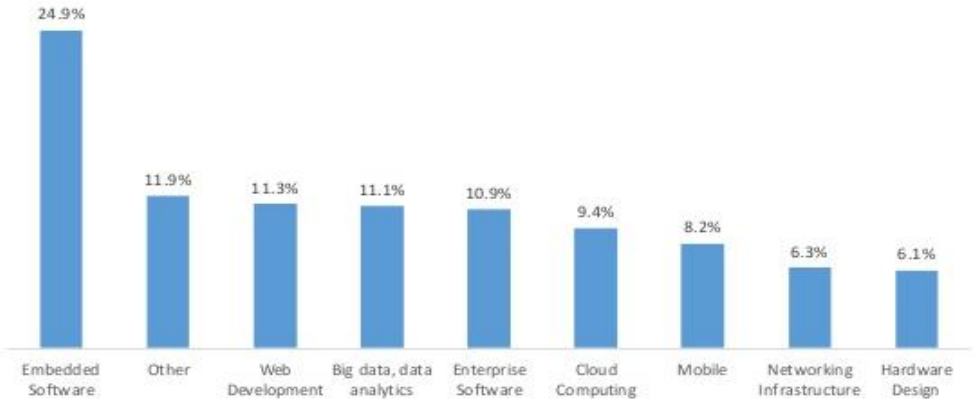


El Open Source es esencial en el IOT

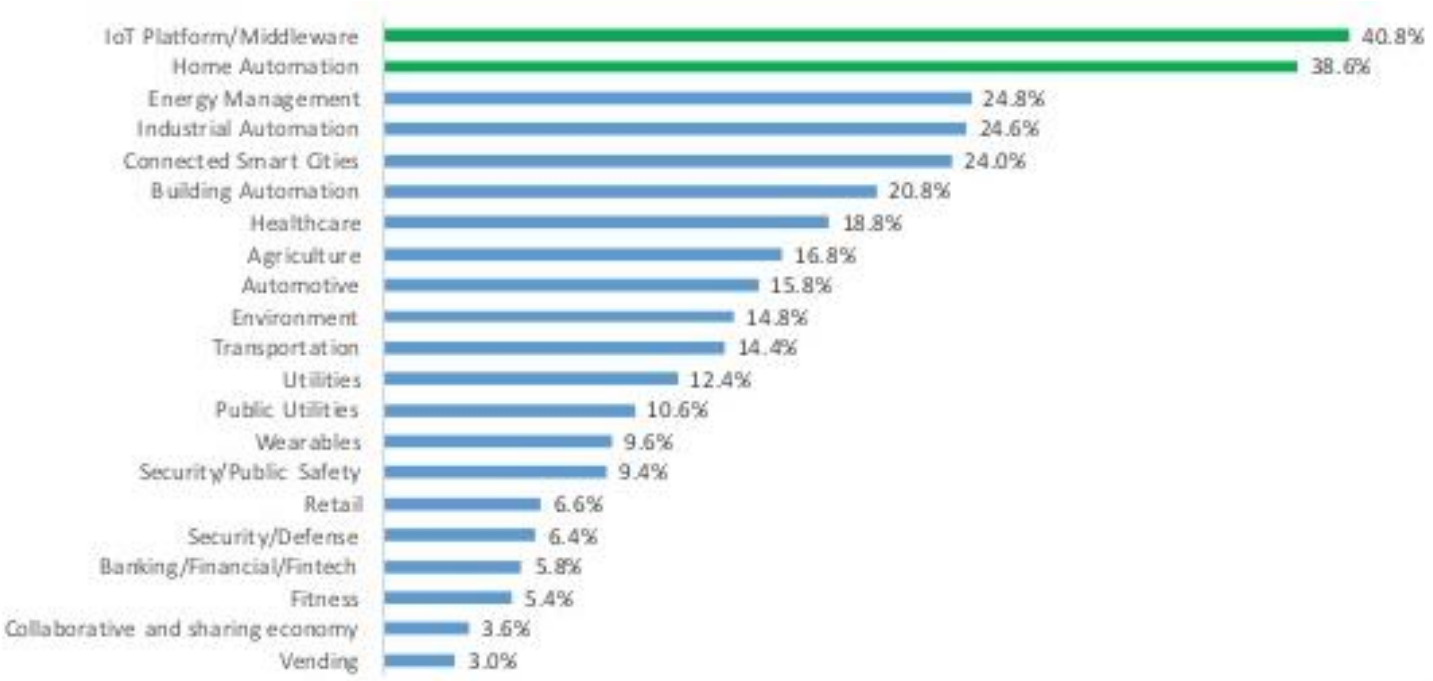
Creo firmemente en que el Open Source es un punto crítico para el éxito de la industria del IOT. El 58% de los encuestados respondieron están muy atentos al Open Source para el IOT. Me agrada el hecho de que Eclipse esté creando una comunidad Open Source para el desarrollo IOT.

Tecnologías en las cuales se aplica el IOT

Las tecnologías en las cuales se aplica el IOT el software embebido descendió de un 35.1% a 24.9%, la sección de “otros” se encuentra con un 11.9% y con un 11.3% el desarrollo web. Las empresas que ya desarrollan e implementan soluciones con IOT solamente subió un 1% alcanzando un 46%, el resto de compañías siguen planeando desarrollar soluciones IOT en los próximos 6 o 18 meses.



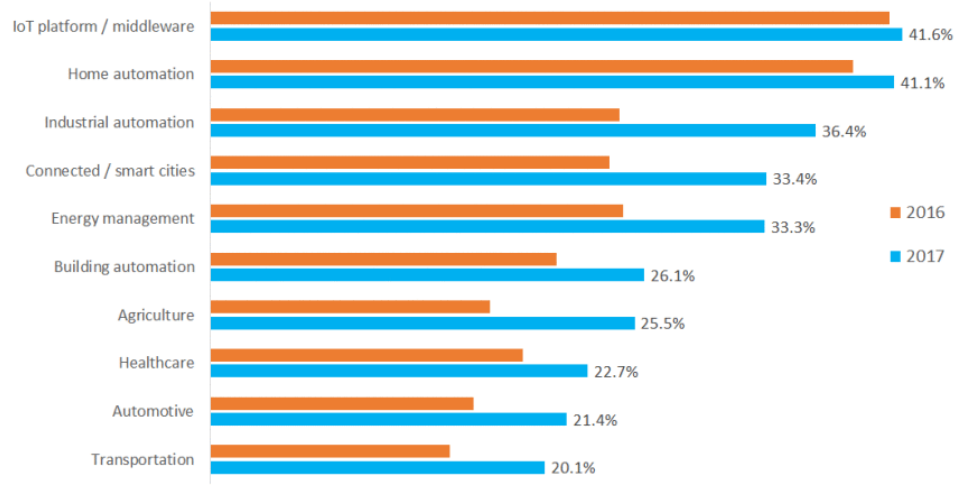
A continuación, se muestran las áreas de soluciones más acudidas:



IOT Developer 2017
Resultados clave

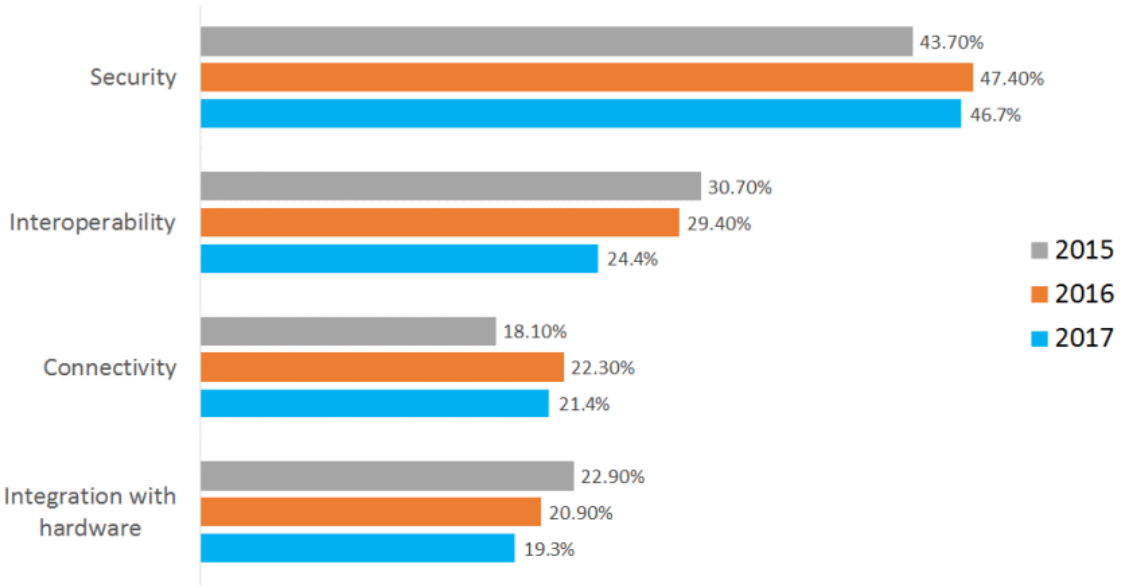
Expansión de la adopción del IOT en las industrias

Las plataformas IOT y la automatización de hogares continúan liderando, no obstante, industrias como la automatización industrial, Ciudades inteligentes y administración de la energía han experimentado un crecimiento entre los años 2015 y 2016.

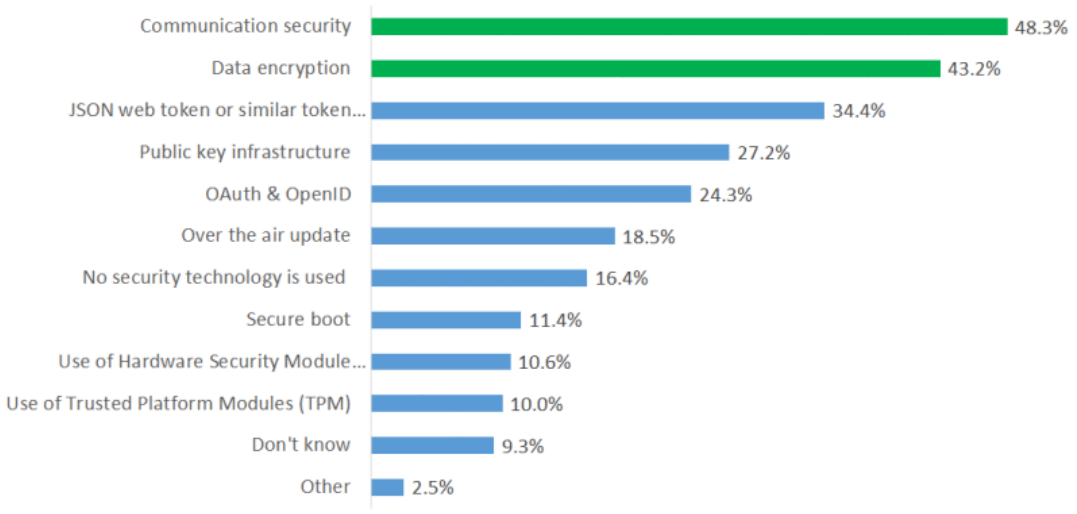


La seguridad continúa siendo la principal preocupación

La seguridad continúa siendo la principal preocupación de los desarrolladores IOT con un 46.7% de los entrevistados indicando esta preocupación. La interoperabilidad con un 24.4% y la conectividad con un 21.4% son las preocupaciones más populares mencionadas. Además, se puede notar que el porcentaje de preocupación por la interoperabilidad ha ido disminuyendo desde el 2015 (30.70%) y 2016 (29.40%) lo cual indica que se ha estado trabajando en dicha preocupación.



A diferencia de las encuestas del año 2015 y 2016 en 2017 se preguntó a los encuestados sobre que tecnologías relacionadas a la seguridad se estaban utilizando para las soluciones IOT. Las dos tecnologías más populares fueron las tecnologías ya existentes, por ejemplo, Las comunicaciones de seguridad (TLS, DTLS) con un 48.3% y la encriptación de datos con un 43.2%. La seguridad orientada a Hardware fueron las menos populares.



Los lenguajes de programación IOT dependen de…

Durante 2015, 2016 y 2017, Java y C continúan siendo los lenguajes de programación principales para el desarrollo IOT junto con el uso de C++, Python y JavaScript. Sin embargo, en 2017 se preguntó a los encuestados el uso de los lenguajes por categorías y al parecer el lenguaje de uso depende del software que se desea desarrollar.

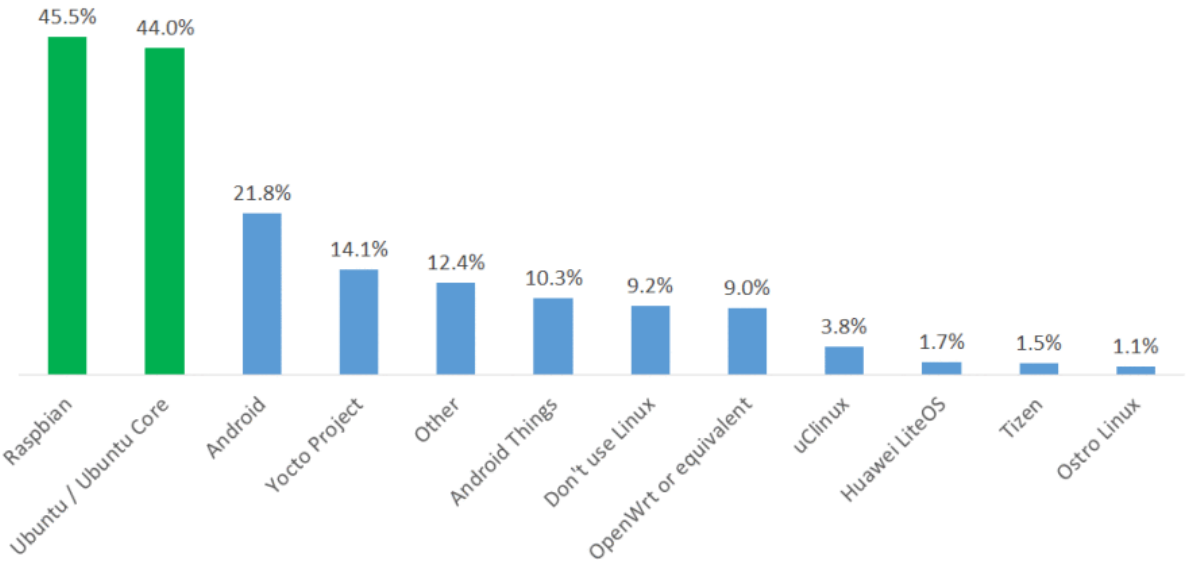
- **En dispositivos restringidos**, se utiliza C con un 56.4%, C++ con 38.3%. Java (21.2%) y Python (20.8%) tienen algo de uso, sin embargo, JavaScript tiene un uso mínimo con un 10.3%
- **Como Gateways IOT**, Los lenguajes de elección son más diversos, Java con un 40.8%, C con 30.4%, Python con 29.9% y C++ con 28.1%. También JavaScript y Node.js tienen un uso, pero un poco más reducido.
- **En plataformas IOT en la nube**, Java con un 46.3% emerge como el lenguaje dominante, JavaScript con 33.6%, Node.js con 26.3% y Python con 26.2%. Sorpresivamente C (7.3%) y C++ (11.6%) descienden significativamente en cuanto a uso en esta categoría

En general, queda claro que el desarrollo para soluciones IOT requiere de una gran diversidad de conocimiento en lenguajes de programación. El lenguaje de programación en realidad depende de lo que se quiere desarrollar.

Linux, el Sistema operativo clave

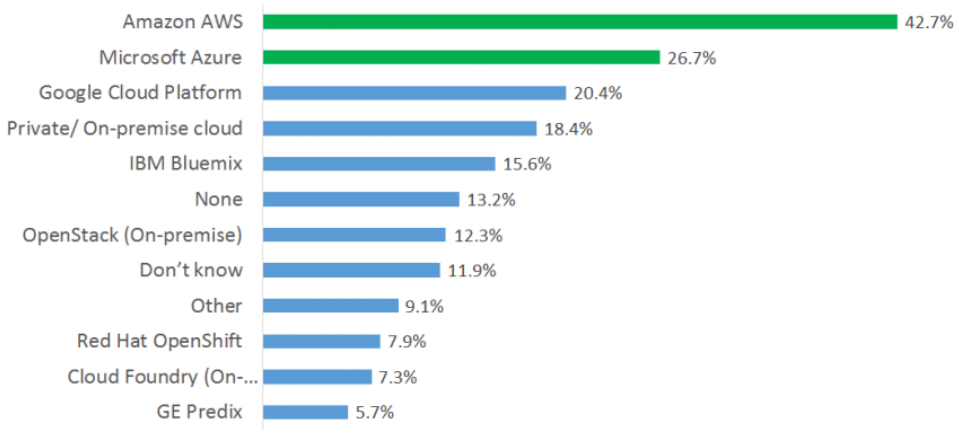
Linux continúa siendo el sistema operativo principal, en 2017 se preguntó a los encuestados si existía algún uso de sistema operativo por categorías, en los dispositivos restringidos Linux lidera con el 44.1% y en segundo lugar se encuentra No OS/ Bar Metal con un 27.6%. En gateways IOT, Linux es muchísimo más usado con un 66.9% de uso mientras que Windows se posiciona como segunda elección con 20.5%.

También se preguntó sobre que distribución de Linux es la más utilizada y se encontró que Raspbian (45.5%) y Ubuntu (44%) son las más utilizadas para IOT. Aunque, si Linux es el sistema operativo predominante ¿Qué sistema operativo sería la alternativo a este? A diferencia de los años pasados en 2017 Windows experimento un gran salto posicionándolo como la segunda opción para IOT.



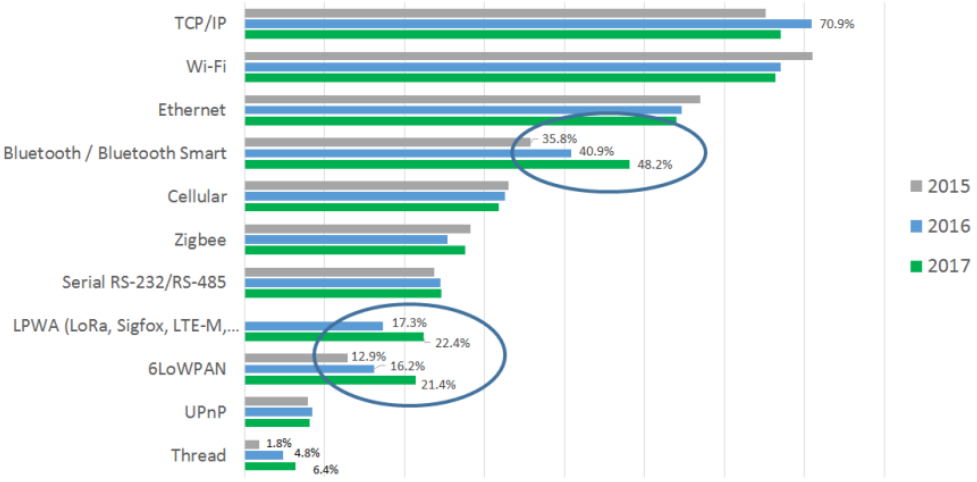
Amazon, Microsoft y Google como plataformas en la nube

Amazon (42.7%) continúa liderando como elección número uno para plataformas en la nube, seguido por Microsoft Azure (26.7%) y Google Cloud Platform (20.4%). Un descenso significativo en 2017 fue la caída del uso de nubes privadas de 34.9% en 2016 a 18.4% en 2017, esto podría ser una señal de que las plataformas en la nube para IOT han madurado más y que los desarrolladores están listos para usarlas.



Bluetooth, los protocolos LPWAN y 6LowPAN son tendencia

En las encuestas pasadas se ha preguntado por los protocolos de conectividad que los desarrolladores utilizan para IOT. La principal respuesta ha sido el protocolo TCP/IP y Wi-Fi, sin embargo, hay un gran número de estándares de conectividad y tecnologías que están siendo desarrolladas para IOT. Basado en los datos obtenidos en la encuesta de 2017 pareciera que Bluetooth/ Bluetooth Smart (48.2%), tecnologías LPWAN (ex Lora, Sigfox, LTE-M) (22.4%) y 6LoWPAN (21.4%) están empezando a ser más usadas por la comunidad de desarrolladores. También el protocolo Thread basado en IPV6 continúa teniendo un "éxito" muy limitado como opción para los desarrolladores pasando de 1.8% en 2015 a 6.4% en 2017.



Resumen general

Durante estos 3 últimos años de encuestas realizadas algunas cosas no han cambiado mucho, otras han cambiado bastante, sin embargo, los resultados muestran patrones en común para los desarrolladores IOT, como la arquitectura IOT de Hardware utilizada, el IDE de uso, las percepciones de los consorcios, la adopción de los estándares, la participación del open source en el IOT y mucho más.

Oracle reconoce a Java como un lenguaje IOT

De acuerdo a un reporte de Oracle, una de las más grandes ventajas de Java es la robustez de código. Explica que el lenguaje C utiliza punteros de manera explícita para hacer referencia a la memoria, sin embargo, todas las referencias a objetos en Java son de manera implícita, es decir, estas referencias a objetos no pueden ser manipulados desde el código de la aplicación. Si esto no fuese así se podrían generar errores potenciales como una violación al acceso de memoria provocando así inevitablemente que una aplicación se detenga prácticamente de forma instantánea.

Mientras que la transferencia de una aplicación programada en C a una nueva plataforma suele ser costosa, quita mucho tiempo y también puede llegar a causar errores, con Java existe la ventaja de que puede ser ejecutado donde sea, por lo tanto, sólo es necesario escribir el código de la aplicación **una vez** para obtener el mismo resultado en distintas plataformas.

Ventajas de utilizar Java para el IOT

- Uno de los usos principales usos para los cuales fue pensado Java fue para ayudar a conectar dispositivos de entretenimiento en el hogar, lo cual hace a este concepto básico de interoperabilidad en la plataforma un encaje perfecto para dispositivos conectados en el ecosistema del IOT.
- Tiene una buena funcionalidad para dispositivos restringidos (circuitos) con un alto nivel de funcionalidad, seguridad, conectividad y escalabilidad en las industrias.
- Es altamente seguro en las actualizaciones de software en el mercado de los dispositivos además de que extiende el ciclo de vida de los dispositivos y permitiendo manejar nuevos servicios remotamente.

- Java cuenta con una gran comunidad de millones de desarrolladores que se encuentran creando aplicaciones Java mientras que otros buscan nuevas formas de lograr un mundo conectado.
- Java ha sido usado desde sus inicios en el sector de la salud con aplicaciones Java embebidas en dispositivos tales como monitoreo de pacientes y cuidado de la salud en el hogar, por otro lado, en la industria automotriz también se utiliza Java como aplicación embebida para la habilitación de sensores y dispositivos IOT.
- El GCF 8 (Generic Connection Framework) con la Access Point API provee de lo último en estándares de seguridad y lo más altos niveles de encriptación en la red y autenticación para asegurar la privacidad de los datos.
- Con Java ME (Micro Edition) aplica lo mismo, una vez que se ha escrito el código puede volver a ser empleado sin necesidad de escribirlo nuevamente para otro dispositivo de hardware.

Es más que notorio que Java siempre fue la primera opción para aquellos desarrolladores que siempre tuvieron en mente un mundo conectado. La tecnología Java siempre ha ofrecido una gran variedad de opciones, seguridad, interoperabilidad, conectividad, reducción de costos, impulsar la innovación, mejorar los servicios de una aplicación, arquitecturas para empresas y para computación en la nube, cabe mencionar que, el IOT sigue en desarrollo y una vez que logre consolidarse podremos estar conectados a través del IOE (Internet Of Everything).

Capitulo III

Introducción

El lugar donde se hará uso de la tecnología Java con el Internet of Things se posiciona en un restaurante, el cual planea abrir nuevas sucursales y ampliarse a diferentes estados de la República Mexicana.

Cada sucursal del restaurante debe tener al menos una PC que este ejecutando la aplicación principal a la cual se van a conectar la cantidad de dispositivos móviles que sean necesarios, estos dispositivos móviles serán usados por los meseros o meseras de dicho restaurante. También deberá de haber como mínimo dos impresoras térmicas conectadas a Internet para impresión de órdenes y tickets, finalmente la sucursal también deberá de contar con un router para llevar a cabo las operaciones en la red. Toda esta infraestructura será implementada con el fin de mejorar el servicio y rapidez con que se atiende a los clientes. Todo ese flujo de información deberá de ser visible para el dueño del restaurante el cual tendrá una aplicación que podrá mostrarle lo que está ocurriendo en su negocio sin necesidad de que se encuentre en el establecimiento.

El dueño del restaurante no desea ver afectada la calidad de conexión a Internet para sus clientes, así que se optó por tener dos routers, uno totalmente dedicado para el tráfico de red de los clientes y otro para el flujo de información de la aplicación.

Propuesta de solución

Lo que se planea es utilizar la tecnología Java para el desarrollo de la aplicación de escritorio y la aplicación móvil más específicamente el desarrollo de la aplicación móvil sería con Android utilizando el IDE Android Studio. Para el backend también se hará uso de un Web Service programado en Java, la infraestructura de la aplicación hará uso de una instancia EC2 de Amazon Web Services la cual contendrá el Web Service anteriormente mencionado, se hará uso de un RDS de Amazon para el almacenamiento de la información de nuestra aplicación, es decir, nuestra base de datos. Se utilizará un “bucket” de S3 para subir archivos tales como imágenes y archivos de oficina. Eventualmente, también se hará uso de Amazon Lambda, un servicio el cual regala el primer millón de peticiones a nuestro servidor de aplicación y el cual sólo cobra por el tiempo en milisegundos que se consume un servicio de nuestra aplicación.

Se planea utilizar Sockets de Java para establecer una comunicación entre la aplicación de escritorio y la aplicación móvil con el fin de intercambiar información y mantener un equilibrio en el flujo de información, el envío de datos a las impresoras térmicas será mediante un escaneo de dispositivos conectados a la red que será realizado desde la aplicación de escritorio, dicho escaneo guardará las direcciones IP de aquellos dispositivos que hayan respondido en cierto puerto estableciendo así un enlace entre los mismos.

Para poder lograr un servicio más rápido a los clientes, una vez que los dispositivos se hayan sincronizado a la aplicación de escritorio los meseros o mesaras podrán hacer uso de la aplicación móvil creando nuevas órdenes desde la misma, las cuales una vez que se ejecute cierta acción en la aplicación, desencadenará una acción la cual se encargará de dividir aquellos productos que sean bebidas o platillos para que en una impresora térmica se impriman las bebidas y en otra los platillos de una orden en particular, acelerando así el proceso de tener que ir a la barra y dejar una nota con la orden nueva e incluso permitiendo al mesero o mesera atender inmediatamente a otro cliente sin perder tiempo.

Servicios de AWS (Amazon Web Services)

Para poder hacer realidad esta aplicación y también ofrecer un alto nivel de robustez en el backend decidimos por usar los servicios de AWS con una instancia C5 las cuales ofrecen un rendimiento más alto debido a que están preparadas para cargas de trabajo intensivo, el modelo de la instancia es una c5.large de 4GB de RAM, con 2 procesadores virtuales y con almacenamiento necesario para instalar un Web Service y dejarlo ejecutando, como dato adicional cabe mencionar que el Web Service será implementado con la tecnología GlassFish Server de Oracle la cual ejecuta Java.

También se incluye el servicio de RDS para almacenamiento de base de datos, este servicio ofrece un límite de almacenamiento gratuito, una vez que se ha sobrepasado el límite se empieza a cobrar el tiempo que se utiliza el servicio.

Por otro lado, tenemos el servicio de S3, este servicio será utilizado para el almacenamiento de archivos, tales como imágenes y archivos de texto que el usuario desee guardar. Este servicio ofrece un 99.99% de durabilidad de los archivos y es usado por empresas líderes en el mercado, consta de subir un archivo al “bucket” de S3 y este te retorna una URL la cual apunta al archivo subido.

Un servicio muy importante a destacar es el servicio Amazon Lambda el cual nos regala el primer millón de peticiones gratis al servidor, una vez sobrepasado ese límite se empezará a cobrar por tiempo de uso, es decir, por el tiempo en el cual se realiza una petición las cuales suelen tardar milisegundos en ejecutarse.

Sabemos que hoy en día y en los últimos tres años Amazon Web Services ha resultado ser una herramienta muy útil cuando de servicios en la nube se habla, además de que también es de las primeras opciones para proyectos de Internet of Things. AWS resulta una mejor opción para aquellas empresas que no desean comprar sus propios servidores y gastar tiempo configurándolos, AWS ofrece los mejores servidores ya optimizados para la demanda que se necesite y a precios muy accesibles.

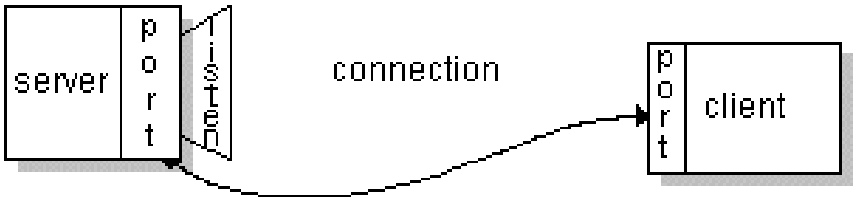
Sockets de Java
¿Qué es un Socket?

Un socket de Java es un end-pont bidireccional, es decir, un puente de comunicación entre dos aplicaciones que están ejecutándose en la misma red. Las clases de Socket son usadas para representar una conexión entre un cliente y un servidor. En el paquete java.net existen dos clases, Socket y ServerSocket que implementan el socket cliente y el socket servidor, respectivamente.

Normalmente, un servidor se ejecuta en una computadora y tiene un socket que está enlazado a un puerto en específico, lo que el servidor hace es esperar, escuchando en su socket a la petición de un cliente. Un socket está enlazada a un puerto de la máquina local, por lo tanto, la capa TCP puede identificar los datos de la aplicación que serán enviados.

Del lado del cliente: En el lado del cliente tenemos que, el cliente, conoce la IP del servidor y el puerto en el cual está escuchando. Para hacer una petición al servidor el cliente intenta conectarse con el servidor con la IP y el puerto específico, además, el cliente también necesita identificarse con el servidor, por lo tanto, se enlaza a un puerto local que se usará durante la conexión. Cabe mencionar que el puerto es asignado por el sistema.

Si todo sale bien, el servidor acepta la conexión y obtiene un nuevo socket enlazado a su puerto local, así como también, su end-point remoto con la IP y puerto local del cliente.



Ahora bien, si en el lado del cliente la conexión es aceptada es cuando un socket es exitosamente creado y el cliente puede empezar a usar el socket para comunicarse con el servidor.

Procedimiento para realizar el escaneo de búsqueda de nodos conectados a la red

Cuando las aplicaciones móviles busquen conectarse al sistema principal deberán de escanear la red en la que se encuentran con el fin de encontrar la IP del sistema principal y que a su vez este les responda con un mensaje y quede a la escucha en un puerto en específico para la recepción de comandos desde las aplicaciones móviles.

Este procedimiento se llevará a cabo con Sockets de Java en donde el sistema principal tomará el papel de Server Socket y las aplicaciones móviles el papel de Client Socket. Cuando un usuario de aplicación móvil abra su aplicación móvil deberá presionar un botón el cual le permitirá sincronizarse y ejecutará el escaneo de IP, lo que este escaneo hará será tomar la IP del dispositivo móvil y extraerá los primeros 3 octetos de la IP, es decir de una IP con la dirección 192.168.60.17 sólo se tomaría 192.168.60, de esta manera se puede deducir que el sistema principal se encuentra en esa red, posteriormente se pasa a una iteración desde 1 hasta 254 para realizar un ping a cada dirección IP en un puerto específico aquella dirección IP que responda al ping se guardará en el dispositivo móvil para futuras conexiones, en caso contrario no se hará nada.

Cabe mencionar que una vez que los dispositivos se enlacen al sistema principal deberán de pasar por un filtro de seguridad el cual autenticará al dispositivo móvil a enlazar, también se registrará al dispositivo con un identificador y clave única para las conexiones que se realicen durante la operación del sistema, operaciones como orden de impresión de tickets, cambios de estados en una orden y ordenes de sincronización.

A continuación, se muestran algunos fragmentos de los códigos para sincronización, revisar si el servidor está disponible, hacer una autenticación con el servidor y un código del servidor personalizado sobre cómo se aceptan las peticiones.

Código de servidor disponible

```
try{
    cs = new Socket();
    cs.connect(new InetSocketAddress(host, Session.PORT), 50);
    final SharedPreferences sharedPreferences = context.getSharedPreferences([redacted], Context.MODE_PRIVATE);
    final Command comando = new Command [redacted];
    final String key = sharedPreferences.getString([redacted], "");
    if (!key.equals("")) {
        comando.setKey(new Key [redacted] key));
    } else
        comando.setKey(new Key [redacted] ""));
    comando.setPassword(Session.PASSWORD);
    serverOutput = new DataOutputStream(cs.getOutputStream());
    serverOutput.writeUTF(new Gson().toJson(comando));

    final InputStream is = cs.getInputStream();
    final DataInputStream dataInputStream = new DataInputStream(is);
    final String message = dataInputStream.readUTF();
    if (message.equals("-1")) {
        return Integer.parseInt(message);
    } else if (message.equals("-2")) {
        return Integer.parseInt(message);
    } else {
        final SharedPreferences.Editor editor = sharedPreferences.edit();
        Session.SERVER_IP = host;
        editor.putString([redacted], host);
        editor.putString([redacted] message);
        editor.apply();
    }
    cs.close();
    return 1;
} catch (IOException ignored) {}
```

El fragmento de código anterior muestra como el dispositivo móvil se conecta con una IP proporcionada y un puerto en específico al cual conectarse, con una espera de 50

milisegundos.

En el desarrollo Android existe una clase llamada SharedPreferences la cual almacena datos en un xml que se guarda en el dispositivo, esta clase permite guardar y recuperar pares de clave-valor de datos primitivos. Si se deseea escribir en el xml de SharedPreferences de debera usar la clase interna de Editor (SharedPreferences.Editor) para obtener una instancia de esta clase se debe usar el método edit() de SharedPreferences, una vez obtenido este método podremos guardar los valores que necesitemos con los métodos proporcionados y finalmente guardarlos con la llamada a un método llamado commit() o apply().

Cuando se obtiene una referencia al objeto SharedPreferences en la aplicación móvil se llama a un método para obtener un identificador único para el teléfono el cual será enviado al servidor. También se crea un comando para el servidor que también será enviado para realizar una operación en específico, además de agregar una contraseña para que pueda ser autenticado en el servidor.

Una vez hecho lo anterior, se utiliza el objeto serverOutput el cual es una instancia de DataOutputStream que recibe un OutputStream del socket cliente (dispositivo móvil) el cual ya contiene la IP y puerto de destino, posteriromente se llama al método writeUTF(String message), este método espera un String como el mensaje que se enviará al servidor, cabe mencionar que para el envío de mensajes se eligió una estructura de objetos JSON (Javascript Object Notation) esta estructura se genera con la libirería Gson que fue escrita en Java por la empresa Google.

Cuando el método writeUTF se ejecuta se envía la información indicada, el servidor recibe esa información la procesa y retorna una respuesta que se ve reflejada en la instancia del objeto InputStream que a su vez necesita de un DataInputStream para acceder a esa información con el método readUTF() que regresa un String de la respuesta del servidor, en el caso del código se revisa si el mensaje retornado fue un -1, un -2 u otro número diferente, para los dos primeros casos significa que de alguna manera el dispositivo móvil no ha sido autorizado y se emprende una acción con base al valor retornado, para el último caso se obtiene la instancia de

SharedPreferences.Editor para guardar la IP y puerto del servidor, finalmente se cierra la conexión con el socket cliente y se retorna un valor para emprender una acción de “éxito”.

Código de autenticación con servidor

```
try{
    cs = new Socket();
    cs.connect(new InetSocketAddress(host, Session.PORT), 50);
    final Command comando = new Command('[redacted]');
    comando.setPassword(Session.PASSWORD);
    serverOutput = new DataOutputStream(cs.getOutputStream());
    serverOutput.writeUTF(new Gson().toJson(comando));

    final InputStream is = cs.getInputStream();
    final DataInputStream dataInputStream = new DataInputStream(is);
    final String message = dataInputStream.readUTF();
    cs.close();
    return (!message.isEmpty())? Integer.parseInt(message): -1;
} catch (IOException ignored) {}
return 0;
```

El código anterior nos muestra un proceso similar al de servidor disponible, se envía un comando nuevamente, únicamente con una contraseña de autenticación para que sea verificada por el servidor, finalmente regresa un mensaje del servidor que es procesado en otro método.

Código de sincronización con servidor

```
final WifiManager wm = (WifiManager) context.getApplicationContext().getSystemService(Context.WIFI_SERVICE);
final int ipAddress = wm.getConnectionInfo().getIpAddress();
final String ip = String.format(Locale.getDefault(), "%d.%d.%d.%d",
        (ipAddress & 0xff), (ipAddress >> 8 & 0xff),
        (ipAddress >> 16 & 0xff), (ipAddress >> 24 & 0xff));
final String[] items = ip.split("\\.");
final int size = items.length - 1;
String ipPattern = "";
for (int i = 0; i < size; i++){
    ipPattern += items[i] + ".";
}
for (int i = 1; i < 256; i++){
    final int response = (streaming)?isReachable(ipPattern + String.valueOf(i), context)
        :authenticate(ipPattern + String.valueOf(i));
    if (streaming){
        if(response == 1){
            return ipPattern + String.valueOf(i);
        } else if (response == -1) {
            return "[redacted]";
        } else if (response == -2) {
            return "[redacted]";
        }
    } else {
        if (response > 0)
            return String.valueOf(response);
        else if (response == -1)
            return String.valueOf(response);
    }
}
return "";
```

En este código, se obtiene una referencia a la clase WifiManager la cual nos permite obtener la dirección IP en un formato de números enteros asignados por el servicio de Wi-Fi. Para obtener la ip del dispositivo se debe tratar con operadores a nivel de bits, se utiliza el método estático de la clase String.format(), este método recibe un objeto Locale el cual representa la localidad en la que se encuentra el dispositivo, esto permite a Java retornar un formato correcto para números, un ejemplo sencillo a continuación:

Locale	Formato de números
Alemania (Alemania)	123.456,789
Alemania (Suiza)	123’456.789
Estados unidos	123,456,789

En la tabla se puede ver como el formato númerico varía dependiendo de las regiones, es por ello que es un parametro requerido en String.format(), el segundo parámetro es un indicador de como debe ser el retorno del String en este caso “%d.%d.%d.%d” indica el número seguido de un punto lo cual nos da el formato dirección IP.

Ahora, tomando el número entero que representa la IP, lo que se hace es aplicar operadores a nivel de bits.

```
final String ip = String.format(Locale.getDefault(), "%d.%d.%d.%d",
        (ipAddress & 0xff), (ipAddress >> 8 & 0xff),
        (ipAddress >> 16 & 0xff), (ipAddress >> 24 & 0xff));
```

En el fragmento de código anterior se muestran los operadores de bits & y >> estos se aplican en el número entero obtenido anteriormente. Suponiendo que el número retornado es 3,232,235,778 lo que se hace con el operador a nivel de bits es tomar el valor decimal y calcularlo a una escala de bits, es decir, se transforma en un número binario, quedando de la siguiente forma:

Decimal: 3,232,235,778

Binario: 11000000 10101000 00000001 00000010

IP: 192.168.1.2

El operador >> es un operador de desplazamiento de bits, lo que se hace en la instrucción (ipAddress >> 24 & 0xff) es desplazar el número binario 24 bits, quedando de la siguiente manera

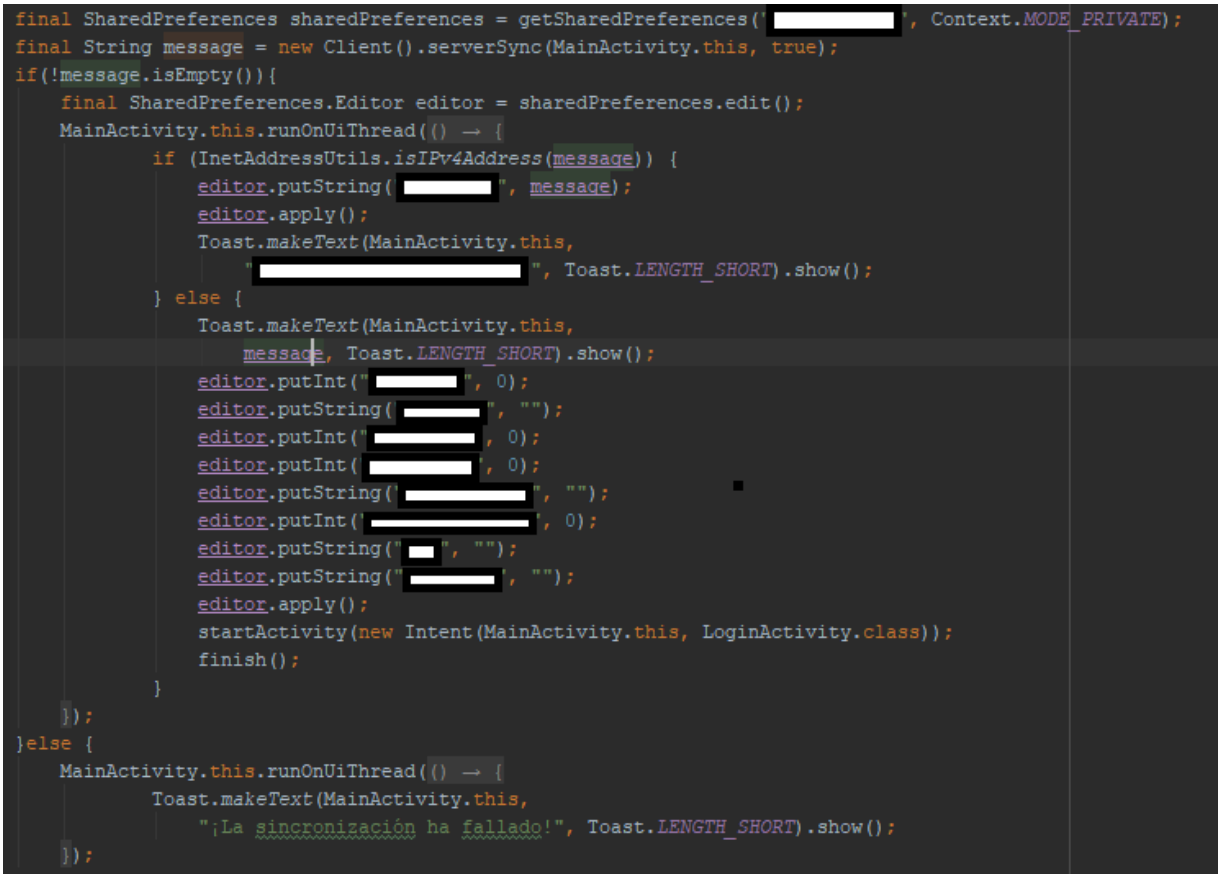
>>>>>>>>>>>>>>>>>>>>>>>> Se recorre 24 bits

00000000 00000000 00000000 11000000

0xff es un número hexadecimal que es expresado como 255, cómo número binario sería 11111111, entonces se aplicar el operador & (AND) con 11000000 y 11111111 ahora bien con base a la tabla lógica del operador AND el resultado de la operación binaria ya en número con base decimal sería 192, este mismo caso se aplicara para el desplazamiento de 16 bits, 8 bits y 0 bits lo que al final nos retornará en el código un string con el formato “%d.%d.%d.%d” que quedaría como 192.168.1.2.

Ya que se ha obtenido la IP del dispositivo se llama al método split() de la clase String el cual retorna un arreglo de String[], este método toma como argumento un separador en este caso toma el separador de punto y nos retorna el arreglo de esta forma {192, 168, 1, 2} y obtenemos los 3 primeros segmentos de la IP, así es como se deduce en que red está el servidor principal siempre y cuando esté conectado a la misma red, entonces es cuando se aplica una iteración 254 veces empezando desde 1, en cada iteración se llama a los métodos previamente explicados y tomando en cuenta un valor booleano se decide que método usar, en cada iteración se pasaría una dirección IP generada por la misma iteración, por ejemplo 192.168.1.1, 192.168.1.2, 192.168.1.3, 192.168.1.n se hace el ping con la dirección y puerto en específico y si se obtiene una respuesta de 1 se retorna la IP en la que coincidió, en los otros casos se retornan valores que tienen que ver con acciones que no son de “éxito”

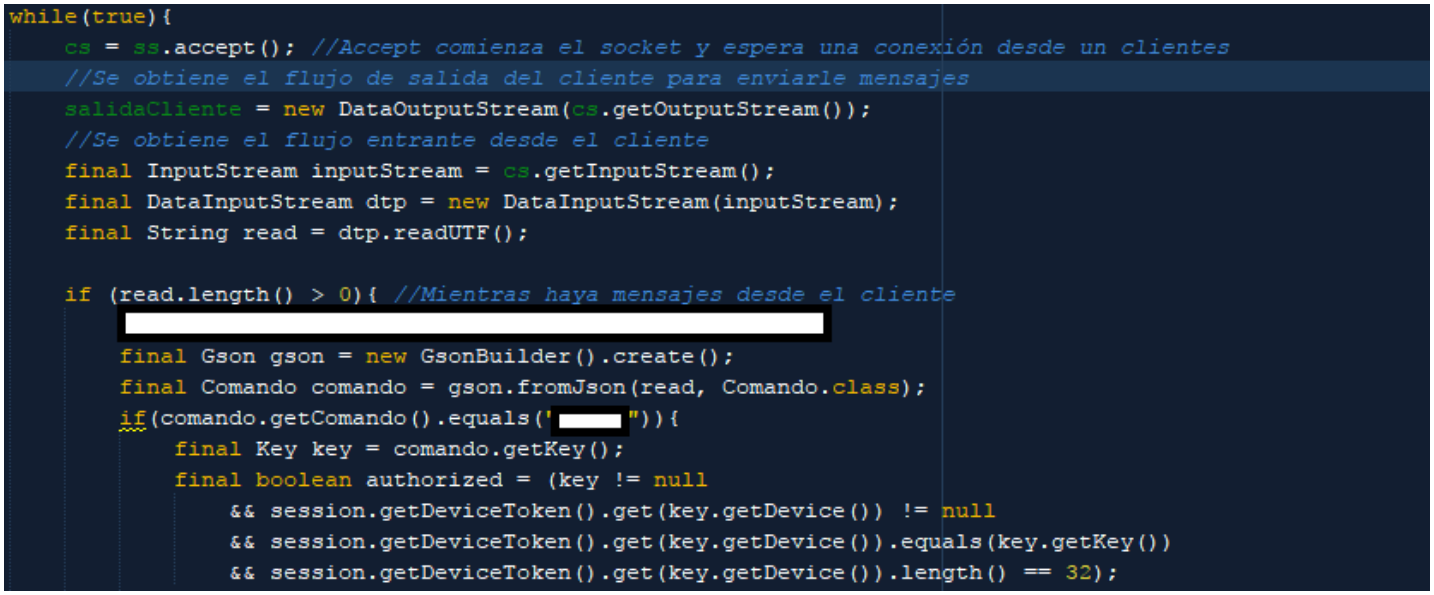
Implementación del código anteriormente explicado



```
final SharedPreferences sharedPreferences = getSharedPreferences([redacted], Context.MODE_PRIVATE);
final String message = new Client().serverSync(MainActivity.this, true);
if(!message.isEmpty()){
    final SharedPreferences.Editor editor = sharedPreferences.edit();
    MainActivity.this.runOnUiThread(() -> {
            if (InetAddressUtils.isIPv4Address(message)) {
                editor.putString([redacted], message);
                editor.apply();
                Toast.makeText(MainActivity.this,
                    "[redacted]", Toast.LENGTH_SHORT).show();
            } else {
                Toast.makeText(MainActivity.this,
                    message, Toast.LENGTH_SHORT).show();
                editor.putInt("[redacted]", 0);
                editor.putString([redacted], "");
                editor.putInt("[redacted]", 0);
                editor.putInt([redacted], 0);
                editor.putString("[redacted]", "");
                editor.putInt("[redacted]", 0);
                editor.putString("[redacted]", "");
                editor.putString("[redacted]", "");
                editor.apply();
                startActivity(new Intent(MainActivity.this, LoginActivity.class));
                finish();
            }
    });
}else {
    MainActivity.this.runOnUiThread(() -> {
            Toast.makeText(MainActivity.this,
                "¡La sincronización ha fallado!", Toast.LENGTH_SHORT).show();
    });
```

En el fragmento de código se puede ver la implementación del método serverSync del objeto Client, como primer parámetro se envía un Contexto y como segundo parámetro si se trata de una sincronización (true) o de una autenticación (false), se revisa que el valor retornado no esté vacío, también se valida que el valor retornado sea realmente una dirección IPv4 para posteriormente ser almacenada de lo contrario la aplicación cargará la vista de Login para autenticarse nuevamente y finalmente, en dado caso de que el mensaje sea vacío se muestra el mensaje de “¡La sincronización ha fallado!”

Código personalizado de autenticaciones en el servidor



```
while(true){
    cs = ss.accept(); //Accept comienza el socket y espera una conexión desde un clientes
    //Se obtiene el flujo de salida del cliente para enviarle mensajes
    salidaCliente = new DataOutputStream(cs.getOutputStream());
    //Se obtiene el flujo entrante desde el cliente
    final InputStream inputStream = cs.getInputStream();
    final DataInputStream dtp = new DataInputStream(inputStream);
    final String read = dtp.readUTF();

    if (read.length() > 0){ //Mientras haya mensajes desde el cliente
        [redacted]
        final Gson gson = new GsonBuilder().create();
        final Comando comando = gson.fromJson(read, Comando.class);
        if(comando.getComando().equals("[redacted]")){
            final Key key = comando.getKey();
            final boolean authorized = (key != null
                && session.getDeviceToken().get(key.getDevice()) != null
                && session.getDeviceToken().get(key.getDevice()).equals(key.getKey())
                && session.getDeviceToken().get(key.getDevice()).length() == 32);
```

El fragmento de código mostrado anteriormente se encuentra en un ciclo infinito, dentro de un bloque while(true) que siempre se estará ejecutando. En la primera línea del bloque while se puede ver la instrucción cs = ss.accept(), este método es el que queda a la escucha de una nueva petición por parte de un cliente. Cuando una petición llega y es aceptada se toma la entrada de datos del cliente con una instancia del objeto DataOutputStream y se toman el mensaje recibido con el método readUTF(), posteriormente se revisa que el mensaje no esté vacío, se instancia un objeto Gson y se toma la estructura de JSON del mensaje para ser serializada a un objeto de clase Comando, este objeto contiene los datos enviados desde el cliente a partir de esa instrucción es cuando entra una serie de evaluaciones para realizar ciertas acciones, cada vez que se entra a una acción se deben autorizar los datos y verificar que el dispositivo haya sido registrado en el sistema principal, una vez que pasa se procede a continuar con la operación.

```
else if(comando.getComando().equals([redacted])){
 if (Session.restaurante.getDispositivos() > 0) {
     final Key key = comando.getKey();
     final boolean authorized = (session.getDeviceToken().get(key.getDevice()) != null
     && session.getDeviceToken().get(key.getDevice()).equals(key.getKey())
     && session.getDeviceToken().get(key.getDevice()).length() == 32);
     if (authorized) { //El dispositivo ya está registrado y puede volver a sincronizarse
         if(comando.getPassword().equals(Session.PASSWORD)){
             if(!key.getDevice().equals("")){
                 if(session.getDeviceToken().get(key.getDevice()) == null){
                     key.setKey(General.toMD5(key.getDevice()));
                     session.getDeviceToken().put(key.getDevice(), key.getKey());
                     salidaCliente.writeUTF(key.getKey());
                 }else{
                     salidaCliente.writeUTF(General.toMD5(key.getDevice()));
                 }
             }
         }
     } else { /*No está registrado, necesita verificarse cantidad de dispositivos
         para continuar con la verificación*/
```

El código anterior muestra la operación para permitir una sincronización con el servidor, se verifica si hay al menos un dispositivo sincronizado, se realiza el método de autenticación y se comprueba si el dispositivo ya está autenticado, si ya ha sido autenticado el dispositivo puede volver a sincronizarse de lo contrario significa que el dispositivo no está registrado en el sistema principal, posteriormente será necesario verificar la cantidad de dispositivos para continuar con la verificación.

```
 } else { /*No está registrado, necesita verificarse cantidad de dispositivos
     para continuar con la verificación*/
     if (session.getDeviceToken().size() < Session.restaurante.getDispositivos()) {
         if(comando.getPassword().equals(Session.PASSWORD)){
             if(!key.getDevice().equals("")){
                 if(session.getDeviceToken().get(key.getDevice()) == null){
                     key.setKey(General.toMD5(key.getDevice()));
                     session.getDeviceToken().put(key.getDevice(), key.getKey());
                     salidaCliente.writeUTF(key.getKey());
                 }else{
                     salidaCliente.writeUTF(General.toMD5(key.getDevice()));
                 }
             }
         }
     } else { //Se alcanzó el limite de dispositivos conectados
         salidaCliente.writeUTF("-2");
     }
 }
```

En el fragmento de código anterior se verifica que la cantidad de dispositivos registrados actualmente en el sistema principal, sea menor de los dispositivos permitidos si es así se procede a verificar que la contraseña sea la correcta, que el nombre del dispositivo no esté vacío y finalmente se verifica que el dispositivo que intente sincronizarse no esté registrado en el sistema si se pasan todos estos filtros se genera un identificador único para el dispositivo, es agregado a la lista de dispositivos conectados y se retorna la llave generada al dispositivo cliente para ser almacenada y usada en peticiones al servidor. En caso contrario se retornará un -2 significando que ya se ha alcanzado el límite de dispositivos, también en caso contrario de que el dispositivo que se intente sincronizar ya exista se retorna de igual manera su identificador para ser almacenado.

Los fragmentos de código mostrados anteriormente sólo muestran una pequeña del uso de Sockets de Java en el sistema y muestra una de las operaciones principales del sistema, sin embargo, los sockets son utilizados para muchas más cosas en la aplicación.

UTF

UTF es el formato de codificación de caracteres que utilizan los sockets de Java para enviar mensajes, cabe mencionar que existen otras maneras de enviar y leer mensajes de un socket de Java, pero writeUTF() y readUTF() son los más utilizados generalmente.

Retrofit

Durante el desarrollo, se utilizó una librería open source llamada Retrofit la cual nos permite realizar peticiones al servidor de AWS, estas peticiones pueden ser asíncronas o síncronas. Todas estas peticiones se hacen utilizando el protocolo HTTP además de que trasforma los objetos Java a JSON permitiendo así su conversión a cualquier otro tipo de objeto en otro lenguaje de programación si llegase a ser necesario.

La librería retrofit permite el uso de anotaciones para los métodos o verbos HTTP que pueden ser útiles para la construcción de una API REST, las anotaciones que pueden ser utilizadas son GET, POST, PUT, DELETE y HEAD.

Llamadas asíncronas

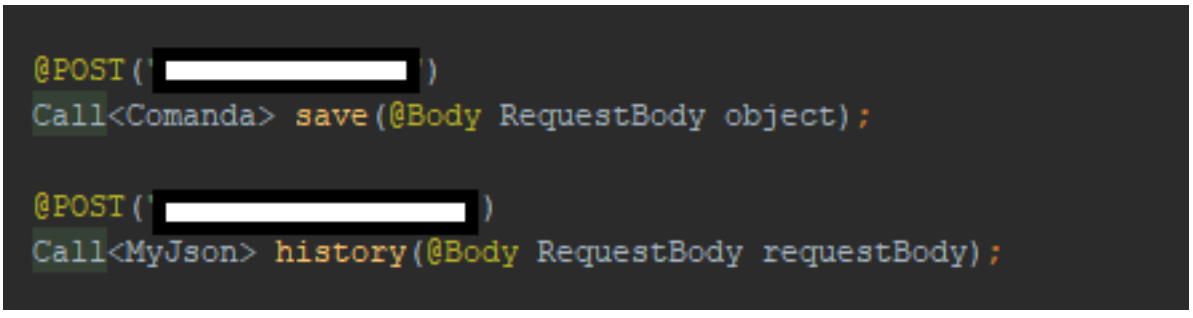
Las llamadas asíncronas son aquellas que no se quedan esperado a una respuesta del servidor, es decir, si se ejecuta una petición al servidor la cual obtenga la lista de platillos de un menú esta llamada se realizará y el resto del código será ejecutado, estas peticiones se ejecutan en un hilo a parte del hilo principal.

Llamadas síncronas

A diferencia de las asíncronas, estás peticiones si esperan una respuesta del servidor, el código que procede de la llamada síncrona si es que existe no es ejecutado hasta haber recibido una respuesta por parte del servidor. En aplicaciones Android si se desea realizar una petición síncrona con retrofit tendría que ser en un hilo, debido a que si no se hace de esta manera podría generarse una excepción de NetworkOnMainThreadException.

Interface de Retrofit

Los métodos para llamar a un servicio de nuestra API (Web Service) se realizan en una interface de Java en la cual además de dar nombre al método y tipo de retorno se agrega el método HTTP y la ruta en la cual se encuentra el servicio.



El fragmento de código anterior muestra dos métodos los cuales pueden ser invocados de manera síncrona o asíncrona, se puede apreciar la anotación @POST sobre el nombre del método, esto indica el método HTTP y dentro de los paréntesis se encuentra el servicio que llama con la estructura {controlador/acción} el método recibe un objeto RequestBody lo cual puede servir para el envío de parámetros al API, en este caso se utilizan objetos JSON para el envío y recepción de datos en el tipo de retorno del método se indica el objeto al cual será convertida la respuesta JSON.

A continuación se muestra la implementación del método history():al

```
final Gson gson = new Gson();
final RequestBody requestBody = RequestBody.create(MediaType.parse("application/json"), gson.toJson([redacted]));
final Call<MyJson> call = orderService.history(requestBody);
call.enqueue(new Callback<MyJson>() {
    @Override
    public void onResponse(Call<MyJson> call, final Response<MyJson> response) {
        if(response.isSuccessful() && response.code() == 200){
            final Gson gson = new GsonBuilder().setDateFormat("yyyy-MM-dd HH:mm:ss").create();
            final MyJson myJson = response.body();
            if(myJson.getTotalCount() > 0) {
                final List<[redacted]> orders = Arrays.asList(gson.fromJson(myJson.getData(), [redacted].class));
                historyAdapter = new HistoryAdapter(orders);
                rvHistory.setAdapter(historyAdapter);
                historyAdapter.notifyDataSetChanged();
                noData.setVisibility(View.INVISIBLE);
                progressDialog.dismiss();
            }else{
                historyAdapter = new HistoryAdapter(new ArrayList<Comanda>());
                rvHistory.setAdapter(historyAdapter);
                historyAdapter.notifyDataSetChanged();
                noData.setVisibility(View.VISIBLE);
                progressDialog.dismiss();
            }
        }else{
            getActivity().runOnUiThread(() -> {
                    Toast.makeText(getContext(), Session.ERROR + response.code(),
                        Toast.LENGTH_SHORT).show();
                    progressDialog.dismiss();
                    noData.setVisibility(View.INVISIBLE);
            });
        }
```

En el código anterior se muestra la implementación de una llamada asíncrona, se crea una instancia del objeto Gson que nos permitirá transformar un objeto Java a un objeto JSON, después se crea una instancia de RequestBody la cual como primer parámetro se indica el tipo de información que se enviará en este caso será un “application/json” y como segundo parámetro se llama al método toJson() de Gson la cual recibe un objeto para transformarlo a una estructura JSON y retornar un String, seguido de esto

con una instancia obtenida de la interface de Retrofit se llama al método y se le pasa el requestBody, este método nos regresará un objeto Call y de este objeto se llama al método enqueue el cual ejecuta la llamada asíncrona en un hilo separado del Main Thread, el método enqueue recibe un Callback con el objeto al cual se transformará la estructura JSON, es decir, de JSON a objeto de Java. De la interface Callback se sobrescriben los métodos onResponse y onFailure, finalmente en el método onResponse se verifica que la respuesta haya sido un código 200 (“OK”) para proceder con el resto del código, en caso contrario sólo se mostrará un mensaje de error en el servidor.

```
@Override
public void onFailure(final Call<MyJson> call, final Throwable t) {
    getActivity().runOnUiThread(() → {
            Toast.makeText(getContext(), Session.FAILURE, Toast.LENGTH_SHORT).show();
            progressDialog.dismiss();
            noData.setVisibility(View.INVISIBLE);
    });
}
```

El método onFailure es llamado sólo en caso de que haya ocurrido un fallo con la conexión, por ejemplo, que no haya acceso a Internet desde la red, en ese caso sólo se muestra un mensaje de fallo en la conexión con el servidor.

```
public static <S> S createService(final Class<S> serviceClass){
    Retrofit retrofit = new Retrofit.Builder()
            .baseUrl(Session.BASE_URL)
            .addConverterFactory(GsonConverterFactory.create())
            .build();
    return retrofit.create(serviceClass);
}
```

El código anterior muestra cómo se crea un servicio con Retrofit, es decir la interface que recibe los datos de nuestra API. Es un método estático que recibe una clase y se adjunta a un objeto retrofit al cual se le indica la URL del servidor del cual se obtendrán los datos, sería una URL base, que con los métodos de las interfaces se concatenan para formar el end-point de donde se extraerán los datos

Notificaciones push

Mientras la aplicación se ejecuta en su entorno, se necesitan de notificaciones que avisen a los usuarios de la misma acerca de lo que está ocurriendo en su entorno. Las notificaciones push toman un papel muy importante en la gran mayoría de las aplicaciones móviles e incluso llegan a ser un requisito muy indispensable para mejorar la experiencia del usuario.

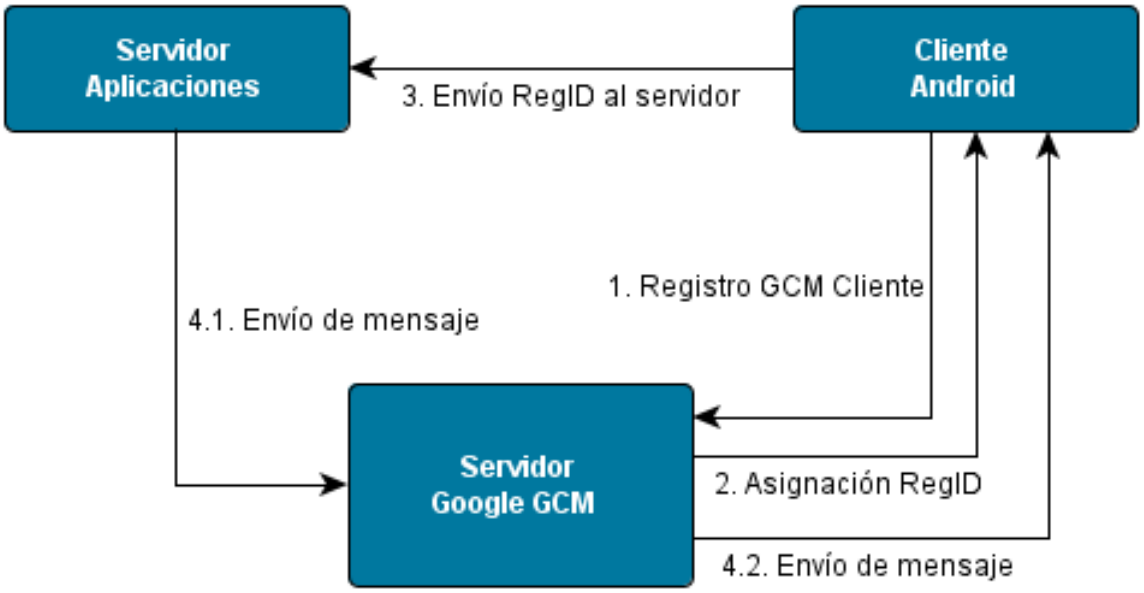
Antes si queríamos estar al día debíamos hacerlo de manera manuel o configurar que de manera automática se hicieran consultas al servidor en busca de mensajes nuevos, fue entonces cuando Blackberry inventó una tecnología a la cual llamo **push** y esta tecnología consistía en que el cliente no debe hacer una consulta al servidor sino más bien el servidor debe hacer la consulta, es decir, en vez de que nosotros preguntemos "¿Hay nuevos mensajes?", es el servidor quien nos dice "Oye tienes un nuevo mensaje".

¿Cómo funcionan las notificaciones push?

Debido a que las notificaciones push son administradas por el servidor, existe un Web Service que se está ejecutando del lado del servidor y que es exclusivo para la recepción y envío de notificaciones push, es decir, tenemos a nuestro cliente la cual es nuestra aplicación de escritorio o aplicación móvil, tenemos nuestro servidor de aplicaciones y nuestro servidor dedicado exclusivamente al manejo de notificaciones push.

El servidor de notificaciones push mantiene una conexión abierta, además de que debe llevar un registro de los dispositivos conectados al mismo, por ejemplo, en una base de datos MySQL. Un dispositivo cliente intenta conectarse con el servidor de notificaciones push, cuando el servidor detecta una nueva conexión se registra al usuario y se genera una clave única para identificarlo, esta clave única es retornada al dispositivo cliente y este mismo envía la clave única al servidor de aplicaciones para ser guardada y utilizada en futuras consultas. Ahora, cada vez que una notificación push deba ser emitida el servidor de aplicaciones enviará la clave única al servidor de notificaciones push para que este se encargue de enviar la notificación al dispositivo con la clave única generada.

A continuación, se muestra un diagrama del envío de mensajes en las notificaciones push:



Cabe mencionar que el envío de notificaciones push en la aplicación es administrado por FCM (Firebase Cloud Messaging) la cual es una mejora de GCM (Google Cloud Messaging) en esencia continua siendo el mismo servicio, en el diagrama se ve claramente como Android se conecta al servidor de Google para recibir una clave única la cual es registrada en el servidor de nuestra aplicación el cual envía el **push** al servidor de notificaciones de Google para que este mismo haga llegar una notificación a la aplicación.

Firebase

Firebase es BaaS (Backend as a Service), actualmente Firebase es más que un BaaS, Firebase puede ser también una **base de datos en tiempo real**, información en tiempo real es lo que actualmente se quiere, estar completamente al día. Convencionalmente, para obtener información de una base de datos necesitan de una petición HTTP para obtener la información, es decir, se debe solicitar la información para poder obtenerla.

Cuando se conecta Firebase a una aplicación, la conexión se está haciendo mediante una petición HTTP, sino más bien mediante un Web Socket, los Web Sockets son mucho más rápidos que una conexión HTTP, no se necesita crear un Web Socket por cada petición, un Web Socket es más que suficiente para llevar a cabo este proceso. Toda la información que se guarda en la base de datos de Firebase es sincronizada automáticamente tan raído como la conexión lo permita.

Firebase también envía información cuando su base de datos sufre cambios, es decir, es actualizada. Por ejemplo, cuando una aplicación cliente actualiza la base de datos de Firebase, todos aquellos clientes conectados a la misma base de datos reciben la actualización casi al instante.

Firebase es **Almacenamiento**. Firebase también ofrece el servicio de almacenamiento, es decir, permite la subida de archivos, comúnmente imágenes, pero puede ser cualquier cosa enviada directamente desde el cliente. Cabe mencionar que Firebase tiene sus propios filtros de seguridad para proteger el “bucket”, además de que también administra detallados permisos de escritura para los clientes que ya hayan sido autenticados por el mismo servicio.

Firebase es **Autenticación**. La autenticación con Firebase se construye con base a un email y una contraseña, además de que también soporta OAuth2 para Google, Facebook, Twitter y GitHub. La autenticación con Firebase también está integrada directamente en su base de datos, por lo tanto, es posible controlar los accesos a la información.

Firebase es **una plataforma completa para aplicaciones**. Se han integrado una gran cantidad de características, todas estas características aplican para el desarrollo IOS y Android, pero no para el desarrollo Web, Firebase sólo está enfocado para ser usado en aplicaciones, algunas de las características son:

- Configuración remota
- Laboratorio de pruebas para aplicaciones
- Reporte de errores, bugs
- Notificaciones (FCM)
- Links dinámicos
- AdMob

Ventajas y desventajas

No todo es color de rosa, bueno con Firebase lo es casi todo, pero también hay que ver las espinas de la rosa

Ventajas

- Autenticación con Email y contraseña, Google, Facebook, Twitter y GitHub.
- Información en tiempo real.
- Una API lista para consumirse.
- Construido con seguridad a nivel de nodos.
- Almacenamiento de archivos respaldado por Google.
- Trata los datos como flujo para construir aplicaciones altamente escalables.
- No es necesario preocuparse por la infraestructura.
- Multiplataforma.

Desventajas

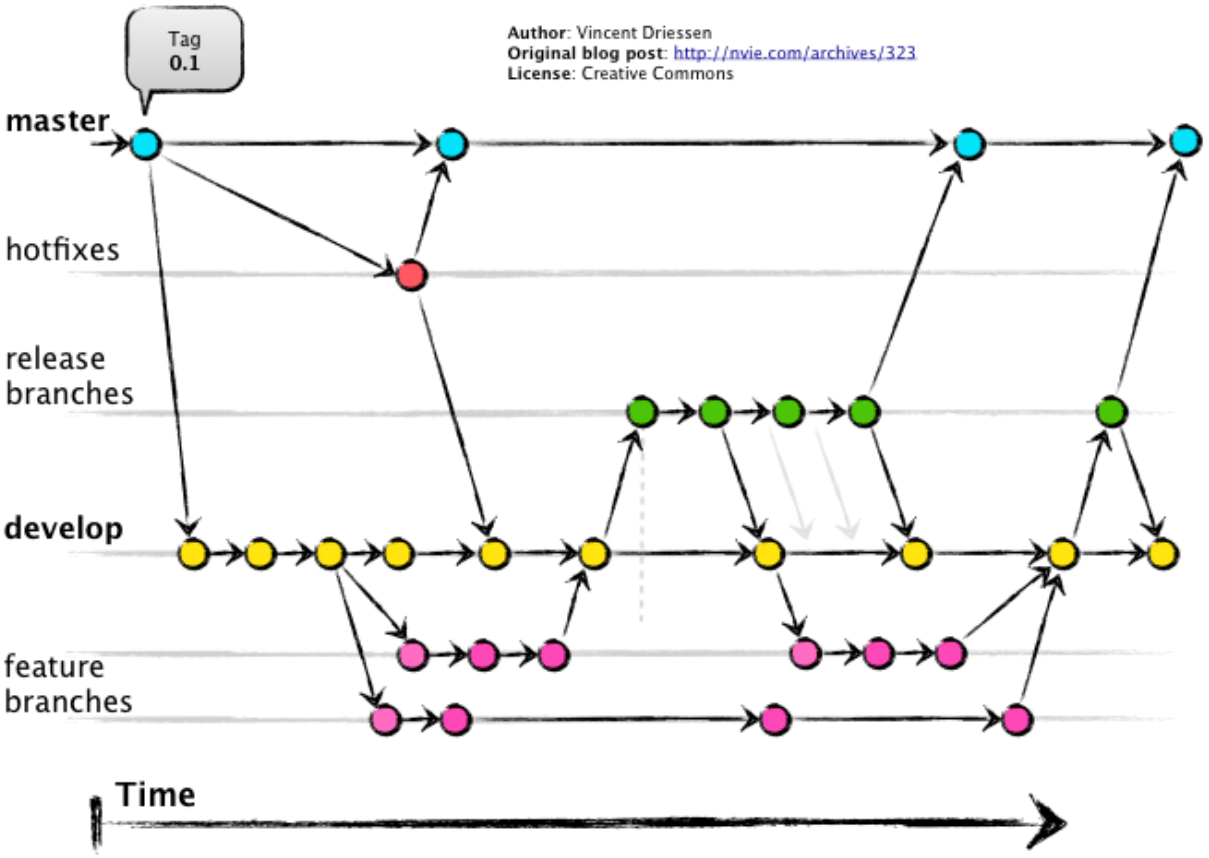
- Métodos de consulta limitadas debido al modelo de flujo de datos de Firebase.
- La escritura de sentencias SQL no es aplicable, todo es mediante métodos ya predefinidos.
- Pruebas limitadas en la nube.
-

OAuth2

El protocolo OAuth2 es un framework de autorización que permite a las aplicaciones obtener acceso limitado a cuentas de usuario mediante un servicio HTTP, cuentas de usuario tales como Facebook, Google, Twitter, GitHub y DigitalOcean, funciona delegando la autenticación del usuario con el servicio que administra la cuenta de usuario y autorizando aplicaciones de terceros para acceder a algunos datos de la cuenta de usuario. Además, OAuth2 provee flujos de autorización para aplicaciones web, de escritorio y dispositivos móviles.

Control de versiones

Un elemento a destacar en el desarrollo de aplicaciones es el control de versiones, es decir, se trata de versionar el código de alguna aplicación en etapas, el versionamiento del código se llevó a cabo con Git, como repositorio de datos se utilizó Bit Bucket , como asistente para la descarga del repositorio o para la subida de versionamientos al repositorio se utilizó el programa SourceTree el cual permite clonar proyectos del repositorio y guardarlos localmente, también tiene todas las herramientas necesarias para un control de versiones. Sin un programa de versionamiento se tendría que ejecutar varios comandos desde la consola para lograr lo que con un botón se lograría un el programa. Como información adicional, cabe mencionar que se utilizó una extensión de Git la cual se llama git-flow, este flujo de trabajo nos permite dividir nuestro código en ramas más específicas.



El diagrama anterior muestra como git-flow funciona, git-flow se divide en dos ramas principales:

- **Master:** En esta rama se hace commit de todo aquello que deba ser subido a producción. Siempre que se incorpore un nuevo código a esta rama es porque se tiene una nueva versión
- **Develop:** En esta rama se encuentra el código que se planea para la siguiente versión planificada del proyecto, es decir, lo que se encuentra en desarrollo o en fase de pruebas.

Además de las dos ramas mencionadas anteriormente también se utilizan las siguientes ramas auxiliares:

- **Feature:** Esta rama se utiliza para desarrollar nuevas características del proyecto que una vez terminadas se mezclan con la rama develop. Un feature siempre se origina a partir de la rama develop y se mezcla siempre con la rama develop.
- **Release:** Esta rama se utiliza para preparar el código hecho en develop antes de mezclarse con producción, es decir, en esta rama se hacen los últimos ajustes y se corrigen los últimos bugs si es que existen para pasarlos a la rama master.
- **Hotfix:** Esta rama se utiliza para corregir errores y bugs en el código de producción, funcionan de manera parecida a una rama reléase sólo que con la diferencia de que un hotfix no es planificado, es decir, un Hotfix siempre es usado para corregir errores y bugs que son fáciles de resolver, por ejemplo, cambiar el color de algún fondo, se tratan de errores y bugs simples.

Todo este proceso de git-flow es administrado por el programa SourceTree lo cual facilita enormemente la implementación de este flujo de trabajo.

Implementación de FCM en la aplicación

Firebase se utiliza en la aplicación como autenticación para ofrecer al usuario la opción de entrar al sistema sin necesidad de registrarse manualmente en la base de datos del sistema, es decir, el usuario se registra en el sistema, pero de una manera más automática.

Otro servicio de Firebase que es utilizado es FCM para notificaciones push, esta acción se ejecuta cuando a un mesero(a) se le notifica que una orden ya está lista para ser entregada, este evento se lleva a cabo cuando la persona que se encuentra en la barra marca una orden como terminada notificando al mesero(a) al cual le corresponde esa orden.

Para implementar la autenticación con Firebase se debe realizar una serie de pasos. Antes de pasar al código se debe registrar el paquete de nuestra aplicación en la consola de Firebase. Una vez hecho el registro Firebase conocerá los dispositivos que tengan la aplicación instalada para poder hacer un tracking de lo que sucede en la aplicación, también Firebase nos proporcionará un objeto JSON el cual contiene una serie de llaves que son importantes para el buen funcionamiento de Firebase en la aplicación, el objeto JSON debe ser incluido en el sistema de archivos de la aplicación.

Una vez que se hayan incluido los archivos y se haya realizado la configuración correspondiente se podrá proceder a implementar Firebase en la aplicación.

```
//FACEBOOK
private CallbackManager callbackManager;
//GOOGLE
private GoogleApiClient mGoogleApiClient;
//AUTH
private FirebaseAuth firebaseAuth;
```

En la imagen anterior se muestran las variables u objetos que se utilizan para realizar la autenticación, el objeto CallbackManager es aquel el cual nos permite autenticarnos con Facebook, GoogleApiClient es un objeto el cual nos permitirá conectarnos con Google y también conectarnos a los servicios de la API de Google, el objeto FirebaseAuth nos permite realizar una autenticación con el servidor de Firebase desde nuestro dispositivo.

```
//FACEBOOK
FacebookSdk.sdkInitialize(getActivity());
callbackManager = CallbackManager.Factory.create();
final GoogleSignInOptions googleSignInOptions = new GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)
    .requestIdToken(getString(R.string.default_web_client_id))
    .requestProfile()
    .requestEmail()
    .build();
//GOOGLE
mGoogleApiClient = new GoogleApiClient.Builder(getActivity())
    .addApi(Auth.GOOGLE_SIGN_IN_API, googleSignInOptions)
    .addOnConnectionFailedListener(this)
    .build();
//AUTH
firebaseAuth = FirebaseAuth.getInstance();
```

En el código anterior se aprecia la instanciación de los objetos previamente descritos, un aspecto a destacar es que para usar el servicio de autenticación con google se debe de indicar en la instanciación que se usará la API de autenticación, entre otros meta datos como requestProfile() y requestEmail(), también se agrega un listener para identificar cuando el objeto GoogleApiClient se haya conectado exitosamente o si hubo un fallo durante la conexión. Al agregar el listener obligatoriamente se tendrá que implementar la interface OnConnectionFailedListener y ConnectionCallbacks, ambas de GoogleApiClient. Una vez implementadas las interfaces se tendrán que sobrescribir algunos métodos.

```
@Override
public void onStart() {
    super.onStart();
    mGoogleApiClient.connect();
}
```

Para que el objeto GoogleApiClient establezca una conexión con el servidor es necesario usar el método connect() el cual recibe una respuesta con los métodos que serán sobrescritos a continuación.

```
//GOOGLE
@Override
/MissingPermission/
public void onConnected(Bundle bundle) {}

@Override
public void onConnectionSuspended(int i) {
    if (getView() != null) {
        ChoiceDialog.showDisclaimer(getContext(), R.layout.dialog_disclaimer,
            "No pudimos obtener información de Google+", true, 0);
    }
    ProgDialog.dismiss();
}
```

Los métodos onConnected y onConnectionSuspended se implementan con la interface ConnectionCallbacks, estos métodos son llamados cuando el proceso del método onConnect() del objeto GoogleApiClient es ejectuado correctamente. En estos métodos se toman acciones las acciones necesarias, en este caso en onConnectionSuspended se muestra un mensaje al usuario si es que la conexión ha sido suspendida.

```java
@Override
public void onConnectionFailed(@NonNull ConnectionResult connectionResult) {
    if (connectionResult.hasResolution()) {
        try {
            connectionResult.startResolutionForResult(getActivity(), ConnectionResult.SIGN_IN_REQUIRED);
        } catch (IntentSender.SendIntentException e) {
            e.printStackTrace();
            mGoogleApiClient.connect();
        }
    } else {
        GoogleApiAvailability.getInstance().getErrorDialog(getActivity(), connectionResult.getErrorCode(), REQUEST_ERROR_RETRY,
                if (getView() != null) {
                    ChoiceDialog.showDisclaimer(getContext(), R.layout.dialog_disclaimer,
                        "No pudimos obtener información de Google+", true, 0);
                }
                ProgDialog.dismiss();
                mGoogleApiClient.disconnect();
        });
        ProgDialog.dismiss();
        mGoogleApiClient.disconnect();
    }
}
```

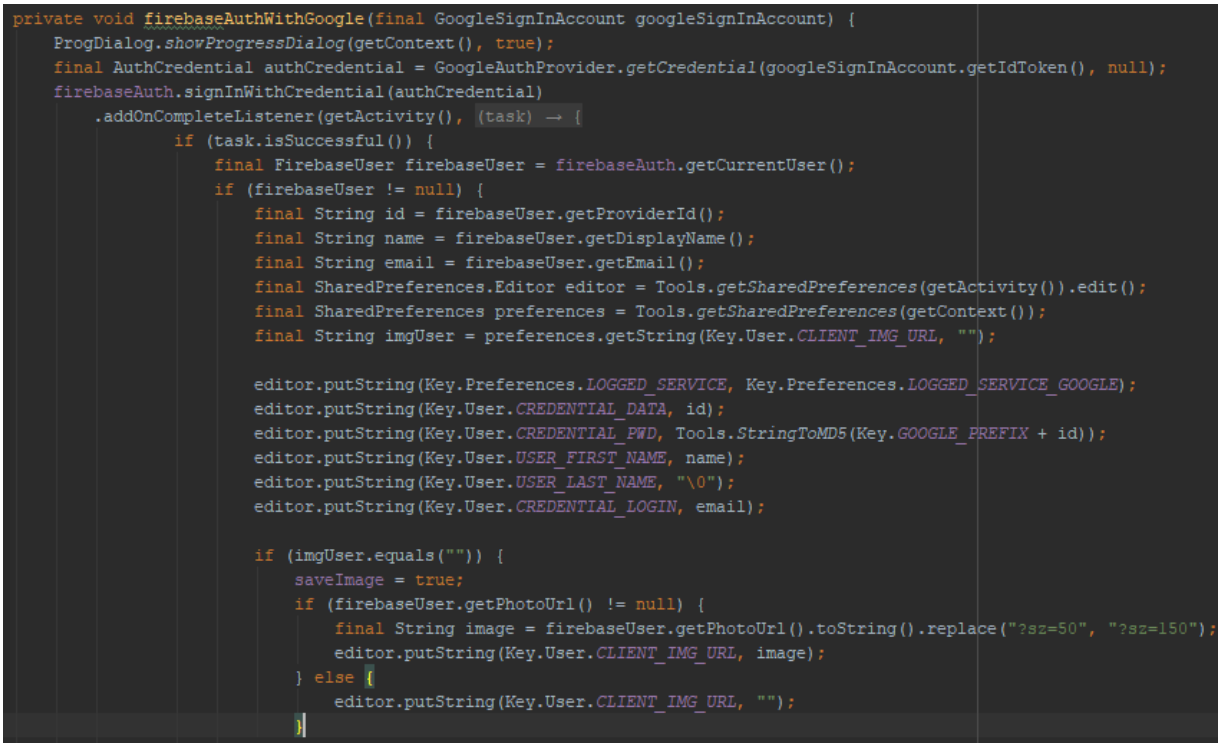
El método onConnectionFailed se ejecuta cuando ha ocurrido un error al intentar conectarse con alguna de las APIS de Google. En este caso se opta por volver a intentar conectarse a los servicios de Google o desconectarse definitivamente.

```java
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    switch (requestCode) {
        //FACEBOOK
        case FACEBOOK_CALLBACK_NUMBER:
            callbackManager.onActivityResult(requestCode, resultCode, data);
            break;
        //GOOGLE
        case REQUEST_ERROR_RETRY:
            if (resultCode == Activity.RESULT_OK) {
                ChoiceDialog.showDisclaimer(getContext(), R.layout.dialog_disclaimer,
                    "No pudimos obtener información de Google+", true, 0);
            }
            break;
        case ConnectionResult.SIGN_IN_REQUIRED:
            //This is when you tried to login but you failed... so try to login again
            mGoogleApiClient.connect();
            break;
        case GOOGLE_SIGN_UP_CODE:
            final GoogleSignInResult googleSignInResult = Auth.GoogleSignInApi.getSignInResultFromIntent(data);
            if (googleSignInResult.isSuccess()) {
                final GoogleSignInAccount googleSignInAccount = googleSignInResult.getSignInAccount();
                firebaseAuthWithGoogle(googleSignInAccount);
            }
            break;
    }
}
```

El fragmento de código anterior es la sobrescritura del método onActivityResult, este método es el producto de la llamada a un método llamado startActivityForResult el cual inicia un nuevo Activity con el fin de retornar una respuesta al Activity que lo ejecutó, es decir de mi Activity A inicio un Acitivty B que después de haber realizado ciertas acciones programadas por el desarrollador o desarrolladora retorna un resultado al Activity A.

Los resultados devueltos por la Activity B se retornan con la clase Intent la cual contiene los datos que se desean retornar, así como también un código de resultado que puede ser cualquier número con el cual se identifica si la operación fue exitosa. Por lo regular, se utilizan constantes para controlar los estados de un resultado. Cuado la Activity A recibe un resultado se llama al método onActivityResult que con un switch se encarga de decidir que acción tomar con base a un requestCode. Esto significa que para autenticarse ya sea con Google o Facebook se debe ejecutar un Activity, pareciera un proceso muy engorroso, sin embargo la librería de autenticación con Facebook y Google ejecutan un Activity que ya está preprogramado y este se encarga de retornar los códigos de estado para tomar la acción correspondiente.

Por ejemplo, cuando se intenta iniciar sesión con Google se llama getSignInResultFromIntent y se le pasa como parámetro un Intent con los datos obtenidos del Activity ejecutado por la librería de Google, entonces se obtiene un resultado para finalmente verificar si la autenticación con Google ha sido correcta y obtener los datos de una cuenta previamente seleccionada en el Activity ejecutado anteriormente. Después de obtener los datos se llama a un método que recibe el objeto GoogleSignInAccount, este método es el que se encargará de realizar la autenticación por medio de Firebase.



```
private void firebaseAuthWithGoogle(final GoogleSignInAccount googleSignInAccount) {
    ProgDialog.showProgressDialog(getContext(), true);
    final AuthCredential authCredential = GoogleAuthProvider.getCredential(googleSignInAccount.getIdToken(), null);
    firebaseAuth.signInWithCredential(authCredential)
        .addOnCompleteListener(getActivity(), (task) -> {
                if (task.isSuccessful()) {
                    final FirebaseUser firebaseUser = firebaseAuth.getCurrentUser();
                    if (firebaseUser != null) {
                        final String id = firebaseUser.getProviderId();
                        final String name = firebaseUser.getDisplayName();
                        final String email = firebaseUser.getEmail();
                        final SharedPreferences.Editor editor = Tools.getSharedPreferences(getActivity()).edit();
                        final SharedPreferences preferences = Tools.getSharedPreferences(getContext());
                        final String imgUser = preferences.getString(Key.User.CLIENT_IMG_URL, "");

                        editor.putString(Key.Preferences.LOGGED_SERVICE, Key.Preferences.LOGGED_SERVICE_GOOGLE);
                        editor.putString(Key.User.CREDENTIAL_DATA, id);
                        editor.putString(Key.User.CREDENTIAL_PWD, Tools.StringToMD5(Key.GOOGLE_PREFIX + id));
                        editor.putString(Key.User.USER_FIRST_NAME, name);
                        editor.putString(Key.User.USER_LAST_NAME, "\0");
                        editor.putString(Key.User.CREDENTIAL_LOGIN, email);

                        if (imgUser.equals("")) {
                            saveImage = true;
                            if (firebaseUser.getPhotoUrl() != null) {
                                final String image = firebaseUser.getPhotoUrl().toString().replace("?sz=50", "?sz=150");
                                editor.putString(Key.User.CLIENT_IMG_URL, image);
                            } else {
                                editor.putString(Key.User.CLIENT_IMG_URL, "");
                            }
```

En la imagen se puede ver el código para autenticarse con Firebase, lo que se realiza primeramente es intentar obtener las credenciales de la cuenta de Google seleccionada proporcionando un token que el servidor de Google procesa para identificar de que usuario se trata y retornar una respuesta.

Con el objeto FirebaseAuth se llama al método signInWithCredential el cual recibe un AuthCredential resultado de la verificación del token de la cuenta de Google, entonces al objeto de FirebaseAuth se le añade un listener para ser notificado cuando las credenciales de Google han sido validadas correctamente, si todo esto se lleva a cabo correctamente se puede acceder al usuario solicitado con Firebase para extraer información, la API de Google sólo ofrece algunos datos, no entrega todos los datos de una cuenta, sólo lo necesario para mostrar en un perfil, estos datos retornados por google pueden ser almacenados en nuestro servidor de aplicaciones, esto puede servir para registrar a un usuario de una manera más automatizada, es decir, sin necesidad de que el usuario deba crearse una cuenta para poder usar la aplicación.

```
                )
            }
            editor.putBoolean(Key.Preferences.LOGGED, true);
            editor.putString(Key.User.CREDENTIAL_PLATFORM, Key.Preferences.LOGGED_SERVICE_GOOGLE);
            editor.apply();
            UserServices.commonLoginUser(getActivity(), LoginFragment.this);
            firebaseAuth.signOut();
            Auth.GoogleSignInApi.signOut(mGoogleApiClient).setResultCallback(new ResultCallback<Status>() {
                @Override
                public void onResult(@NonNull Status status) {}
            });
            ProgDialog.dismiss();
        } else {
            Tools.logOut(getActivity());
            ProgDialog.dismiss();
            if (getView() != null) {
                ChoiceDialog.showDisclaimer(getContext(), R.layout.dialog_disclaimer,
                        "No pudimos obtener información de Google+", true, 0);
            }
            Toast.makeText(getActivity(), "Authentication failed.",
                Toast.LENGTH_SHORT).show();
        }
    }
});
```

Lo que procede es guardar los datos y realizar un signOut(), pero ¿Por qué cerrar sesión cuando apenas se ha iniciado? El motivo es debido a que si no cerramos sesión no estará disponible la opción para iniciar sesión nuevamente con una cuenta diferente y siempre se iniciará sesión con la cuenta previamente elegida, aunque en realidad esto depende de la lógica de la aplicación.

En caso contrario de que la cuenta no haya sido validada, se hace un logOut para salir de la aplicación y se muestran los mensajes correspondientes para notificar de un fallo.

Ahora bien, una vez visto el proceso de autenticación con Firebase podemos ver la implementación de las notificaciones push con FCM.

```
@Override
public void onMessageReceived(RemoteMessage remoteMessage) {
    displayNotification(remoteMessage.getNotification(), remoteMessage.getData());
    sendNewPromoBroadcast(remoteMessage);
}
```

Para poder recibir notificaciones push se debe sobrescribir el método onMessageReceived, este método se sobrescribe heradando de la clase FirebaseMessagingService.

El método tiene como parámetro un objeto RemoteMessage, este objeto contiene el mensaje enviado desde el servidor en una estructura creada por nosotros o con la estructura de texto plano con la cual retorna el servidor.

```
private void displayNotification(RemoteMessage.Notification notification, Map<String, String> data) {
    String message = "";
    if (notification != null) {
        message = notification.getBody();
    } else if (data != null) {
        message = data.get[redacted];
    }
    [redacted]
    //Log.i("Notification", message);
    if (message != null && message.contains("|")) {
        [redacted]
        message = message.substring(message.indexOf("|") + 1, message.length());
    }

    Intent intent = new Intent(this, SplashActivity.class);
    intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
    PendingIntent pendingIntent = PendingIntent.getActivity(this, 0 /* Request code */, intent, PendingIntent.FLAG_ONE_SHOT);

    Uri defaultSoundUri = RingtoneManager.getDefaultUri(RingtoneManager.TYPE_NOTIFICATION);
    NotificationCompat.Builder notificationBuilder = new NotificationCompat.Builder(this)
            .setSmallIcon(R.drawable.icon).setLargeIcon(BitmapFactory.decodeResource(getResources(), R.drawable.icon))
            .setContentTitle[redacted]
            .setContentText(message)
            .setAutoCancel(true)
            .setSound(defaultSoundUri)
            .setContentIntent(pendingIntent);

    NotificationManager notificationManager = (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);
    notificationManager.notify(0, notificationBuilder.build());
}
```

En el método displayNotification lo que se hace es procesar el mensaje recibido desde el servidor con el fin de mostrar una notificación en el teléfono para que el usuario pueda emprender una acción con base a los datos recibidos en el mensaje.

Lo que se hace es verificar si el objeto Notification es diferente de nulo, debido a que la notificación push puede venir en una estructura especial la cual se debe procesar, de lo contrario se obtiene un objeto con una llave para ser tratado de una forma distinta.

Una vez pasado los filtros se prepara un PendingIntent para ejeutar una Activity con los datos obtenidos para cuando el usuario haya decidido abrir la notificación.

También se añade la URL de un sonido de notificación por default para avisar al usuario, posteriormente se construye la notificación a mostrar utilizando la clase NotifictionCompar.Builder, esta clase se instancia pasando el contexto actual de la aplicación, se le agrega un ícono a la notiicación, un título, un texto el cual es el mensaje a mostrar en la notificación, un sonido y un Intent para pasar los datos al Activity a ejecutar cuando se decida abrir la notificación.

Finalmente, se llama a la clase NotificationManager que a su vez llama al método notify el cual lanza la notificación para ser mostrada en el celular.

Raspberry PI

Raspberry PI es una placa de computadora reducida la cual suele ser utilizada para el desarrollo de aplicaciones IOT, esta placa esta compuesta por un procesador Broadcom, memoria RAM, una GPU, puertos USB, HDMI, Ethernet, 40 pines GPIO, un conector para cámara y una Memoria SD o MicroSD.

El Raspberry PI fue utilizado durante el desarrollo de la aplicación, esta placa fue colocada en contenedores de alimentos y conectada a la nube para emitir reportes en tiempo real de lo que está sucediendo. Con ayuda de sensores programados por terceros se logró medir el nivel de temperatura que estos alcanzan para ser reportados directamente a una aplicación en el cual se pueden controlar estos niveles de temperatura remotamente o incluso apagarlos.

Debido a que Raspberry PI es una computadora se puede instalar Java, más específicamente la versión embedded de Java que pesa alrededor de 32MB aunque algunas placas Raspberry PI ya tienen el JDK de Java instalado.

Capitulo IV

Introducción

Después de haber investigado más acerca de la tecnología Java como uso en el IOT, me he dado cuenta de que es una tecnología muy utilizada para el desarrollo de soluciones, no cabe duda alguna de que Java siempre estuvo en la mente de la gran mayoría de los desarrolladores. Lo que se puede lograr con Java es simplemente increíble, tener todo un ecosistema de dispositivos conectados a Internet ejecutando Java vuelve a todo un ambienete más agradable tanto como en tiempo de ejecución y en desarrollo.

Análisis de datos

A pesár de que la tecnología Java es la más utilizada para soluciones IOT, durante el desarrollo de la aplicación se pensó en implementar de otra manera las funciones descritas durante el desarrollo de la misma.

Sockets y Notificaciones push

Durante la implementación de los Sockets con Java para el envío de información, se pensó que de igual manera se podría realizar con notificaciónes push, en efecto es mejor realizarlo con notificaciones push, aunque sería mejor aún implementarlo de las dos formas.

Mientras que con un Socket de Java necesitamos de la IP del servidor para conectarnos e iniciar el envío de mensajes surge el problema de obligar al usuario a mantener su computadora y dispositivos móviles conectados a la misma red, ya que si en algún momento el servidor llegase a cambiar de red la implementación con Sockets ya no sería la más factible.

Con las notificaciones push existe la ventaja de poder hacer llegar una notificación a

un dispositivo sin importar en que red se encuentra, el único requisito es que el dispositivo tenga un conexión estable a Internet, nuevamente se contradice un poco porque con un Socket de Java sólo se necesita de la red sin importar que haya una conexión a Internet, es decir, toda la transmisión de datos se lleva a cabo en la red local. De cualquier manera la aplicación necesita de Internet para funcionar, así que en una etapa del desarrollo de la aplicación se llegó a pensar en implementar notificaciones push y sockets, así cuando uno de ellos llegase a fallar se tendría otra opción para mantener funcionando la aplicación, siempre dando prioridad a las notificaciones push.

Otro punto a destacar es que con las notificaciones push de FCM son más fáciles de implementar debido a que ya es toda una infraestructura que hace el trabajo por nosotros, por otro lado con los Sockets de Java se debe programar todo un protocolo para hacer una solución más completa y personalizada de la aplicación.

Tecnología de Backend

Se debatió bastante sobre que usar para el Backend de la aplicación al final se decidió usar Java, sin embargo, hubo un candidato muy defendido, su nombre es Node.js seguido de C#. El lenguaje C# fue principalmente debatido por el hecho de que es la tecnología Backend principal del equipo de desarrollo, por otro lado, Node.js ha sido una tendencia actualmente e incluso se ha llegado a demostrar que es un 20% más rápido y eficaz que Java, es decir, 20% más rápido ante un lenguaje el cual tiene su propia máquina virtual contro uno el cual es interpretado.

Las principales ventajas de Node.js sobre Java son:

- Proceso de construcción simple: En Java existen diferentes formas de construir o ensamblar el código, por ejemplo Maven y Ant revolucionaron la manera en la que se construye y se obtienen los recursos de una aplicación, de cualquier manera eventualmente se vuelve tedioso, con Node.js sólo basta con guardar un archivo.
- Consultas a base de datos: Mientras que en Java se debe escribir la consulta

SQL, con Node.js sólo se llaman a métodos que se encargan ya sea de insertar, actualizar, eliminar, consultar, etc. Lo cual permite a los desarrolladores de Node.js dejar de preocuparse por problemas de sintáxis.

- Objetos JSON: Cuando una consulta a la base de datos retorna un valor, con Java se debe tratar ese resultado por varias fases ya sea transformarlo a un POJO, utilizar Hibernate, entre otras cosas y configurar todo esto puede llevar un tiempo considerable. Por otro lado, los objetos JSON son una parte natural de Node.js. Cabe mencionar que hoy en día el uso de objetos JSON es muy común incluso en aplicaciones Java, pero un JSON es parte de Javascript y no se necesita de una librería para convertirlo simplemente está ahí.
- Velocidad: La comunidad de desarrolladores se ha sorprendido con la velocidad de Node.js. Sólo nos proecupamos por escribir el código necesario y Node.js se encarga de optimizarlo a lo más rápido posible.

Por estas y otras cuantas razones es por lo cual se tuvo a Node.js muy en cuenta al momento de pensar en la tecnología de Backend.

Amazon

Una de las mejores desiciones fue Amazon debido a que provee una infinidad de servicios de Backend los cuales agilizan bastante el proceso de desarrollado de una aplicación ahorrando mucho tiempo y sólo enfocándose en dar un buen servicio. Cabe mencionar que en los últimos tres años Amazon ha crecido enormemente llegando a ser una de las herramientas principales de servicios de Backend en las empresas.

Sin AWS se habría tenido que diseñar una infraestructura más compleja la cual llevaría a un tiempo de desarrollo más lento, además de que también aumentaría el costo del proyecto. Se hubiese tenido que configurar un servidor para bases de datos, uno para el Web Service, Otro para el servicio de notificaciones push, entre otros cuantos más.

Además, AWS ya ofrece una alta seguridad en sus servidores por lo que sería aún más conveniente que definir la seguridad en servidores propios, cabe mencionar que los servidores de AWS están optimizados por expertos a su máximo potencial en su

respectivo ámbito, es decir, si se renta el servicio de b.ase de datos de AWS el servidor ya estará optimizado para tratar exclusivamente con bases de datos

Capítulo V

Conclusiones

Actualmente vivimos en un mundo globalizado donde la tecnología juega un papel muy importante en el desarrollo de cada uno de los países. Internet no es una tecnología nueva, es una tecnología que existe desde ya hace varios años y que en la última década se ha empezado a explotar y sacar su máximo potencial para implementar el Internet en aparatos a los cuales jamás se pensó que podrían estar conectados a Internet.

La comunicación es un factor muy importante e Internet nos provee esa facilidad de comunicación, que mejor que estar fuera de tu hogar y saber qué es lo que está ocurriendo, ajustar la temperatura de tu hogar remotamente emitiendo una señal a un termómetro conectado a Internet o incluso un hogar lo suficientemente inteligente como para que detecte que tan cerca estamos de nuestro hogar y que comience a preparar una taza de café y encienda la televisión en nuestro canal favorito.

Sin duda alguna la aparición de Internet fue un detonante para la sociedad y lo seguirá siendo durante mucho tiempo, pronto viviremos en una sociedad totalmente conectada a través de Internet y para ello se necesita de una tecnología capaz de soportar toda esta carga y que al mismo tiempo ofrezca seguridad, rapidez y un entorno agradable de ejecución, esta tecnología podría ser Java la cual es un candidato perfecto para hacer realidad este nuevo mundo conectado y que incluso hoy en día se aplica en el desarrollo IOT.

Referencias bibliográficas

https://www.slideshare.net/IanSkerrett/iot-developer-survey-2015

https://www.slideshare.net/IanSkerrett/iot-developer-survey-2016

https://www.slideshare.net/IanSkerrett/iot-developer-survey-2017

https://www.javaworld.com/article/2848210/java-me/java-me-8-and-the-internet-of-things.html

http://www.oracle.com/technetwork/articles/java/java-maker-iot-2214499.html

Apendices y/o Anexos

https://docs.oracle.com/javase/tutorial/networking/sockets/definition.html

https://docs.oracle.com/javase/tutorial/i18n/locale/create.html

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/op3.html