



ESPE

UNIVERSIDAD DE LAS FUERZAS ARMADAS
INNOVACIÓN PARA LA EXCELENCIA

**DEPARTAMENTO DE ELÉCTRICA, ELECTRÓNICA Y
TELECOMUNICACIONES**

**CARRERA DE INGENIERÍA EN ELECTRÓNICA Y
TELECOMUNICACIONES**

**TRABAJO DE TITULACIÓN, PREVIO A LA OBTENCIÓN DEL
TÍTULO DE INGENIERA EN ELECTRÓNICA Y TELECOMUNICACIONES**

**TEMA: DESARROLLO DE UNA ARQUITECTURA CROSS-DEVICE
PARA CLOUD COMPUTING APLICADA A UN SISTEMA EHEALTH**

AUTOR: ALAVA BRAVO, MARÍA JOSÉ

DIRECTOR: Ing. ALULEMA FLORES, DARWIN OMAR MSc.

SANGOLQUÍ

2019

Certificado del Director



**DEPARTAMENTO DE ELÉCTRICA, ELECTRÓNICA Y
TELECOMUNICACIONES**

**CARRERA DE INGENIERÍA EN ELECTRÓNICA Y
TELECOMUNICACIONES**

CERTIFICACIÓN

Certifico que el trabajo de titulación, **“DESARROLLO DE UNA ARQUITECTURA CROSS-DEVICE PARA CLOUD COMPUTING APLICADA A UN SISTEMA EHEALTH”** fue realizado por la señorita **Alava Bravo, María José** el mismo que ha sido revisado en su totalidad, analizado por la herramienta de verificación de similitud de contenido; por lo tanto cumple con los requisitos teóricos, científicos, técnicos, metodológicos y legales establecidos por la Universidad de Fuerzas Armadas ESPE, razon por la cual me permito acreditar y autorizar para que lo sustente públicamente.

Sangolquí, 25 de Enero del 2019

Firma

Ing. Darwin Omar Alulema Flores MSc.

C. C. 1002493334

Autoría de Responsabilidad



**DEPARTAMENTO DE ELÉCTRICA, ELECTRÓNICA Y
TELECOMUNICACIONES**

**CARRERA DE INGENIERÍA EN ELECTRÓNICA Y
TELECOMUNICACIONES**

AUTORÍA DE RESPONSABILIDAD

Yo, **Alava Bravo, María José**, declaro que el contenido, ideas y criterios del trabajo de titulación: ***Desarrollo de una arquitectura cross-device para cloud computing aplicada a un sistema eHealth*** es de mi autoría y responsabilidad, cumpliendo con los requisitos teóricos, científicos, técnicos, metodológicos y legales establecidos por la Universidad de Fuerzas Armadas ESPE, respetando los derechos intelectuales de terceros y referenciando las citas bibliográficas.

Consecuentemente el contenido de la investigación mencionada es veraz.

Sangolquí. 25 de Enero del 2019

Firma

María José Alava Bravo

C.C.: 1722500061

Autorización

**DEPARTAMENTO DE ELÉCTRICA, ELECTRÓNICA Y
TELECOMUNICACIONES**

**CARRERA DE INGENIERÍA EN ELECTRÓNICA Y
TELECOMUNICACIONES**

AUTORIZACIÓN

*Yo, **Alava Bravo, María José** autorizo a la Universidad de las Fuerzas Armadas ESPE publicar el trabajo de titulación: **Desarrollo de una arquitectura cross-device para cloud computing aplicada a un sistema eHealth** en el Repositorio Institucional, cuyo contenido, ideas y criterios son de mi responsabilidad.*

Sangolquí, 25 de Enero del 2019

Firma

.....

María José Alava Bravo

C.C.: 1722500061

DEDICATORIA

“Para desarrollar tu máximo potencial debes ser consciente de que no hay atajos; para tener éxito hay que esforzarse, perseverar y luchar”

Este trabajo se lo dedico especialmente a mis padres, Derlin y Jackeline, gracias a su apoyo hoy en día estoy alcanzando este logro importante, gracias a su guía y consejos toda la vida. Con su ejemplo me enseñaron que para cumplir metas en la vida hay que luchar, trabajar y esforzarse por conseguirlas.

María José Alava B.

AGRADECIMIENTOS

Agradezco a Dios por darme la bendición de poder alcanzar una meta importante en mi desarrollo profesional.

Agradezco a mi familia por ser un apoyo incondicional en todo el camino que he recorrido durante mi vida estudiantil y personal. A mis amigos por ser compañeros y compañeras increíbles y con los que hemos pasado dificultades pero también alegrías y risas. Y a mi novio Javier Viteri, por creer en mí y alentarme en cada momento difícil y motivarme a culminar esta etapa.

Finalmente agradezco a todos mis profesores y a mi director Ing. Darwin Alulema por guiarme exitosamente en el desarrollo de éste trabajo.

María José Alava B.

ÍNDICE DE CONTENIDOS

DEDICATORIA.....	IV
AGRADECIMIENTOS	V
ÍNDICE DE CONTENIDOS	VI
ÍNDICE DE FIGURAS	IX
ÍNDICE DE TABLAS	XIII
RESUMEN	XIV
ABSTRACT	XV
CAPÍTULO 1: DESCRIPCIÓN DEL PROYECTO	1
1.1. ANTECEDENTES.....	1
1.2 JUSTIFICACIÓN E IMPORTANCIA	3
1.3. ALCANCE.....	5
1.4. OBJETIVOS.....	6
1.4.1. Objetivo general.....	6
1.4.2. Objetivos específicos	7
1.5. ESTADO DEL ARTE	7
CAPÍTULO 2: MARCO CONCEPTUAL.....	14
2.1. DESARROLLO DE SOFTWARE DIRIGIDO POR MODELOS	14
2.1.1. MDA	16
2.1.2. Niveles de abstracción.....	18
2.1.3. Lenguaje de modelado	19
2.2. MODELOS DE SERVICIO CLOUD	23

2.2.1. SaaS.....	23
2.2.2. PaaS.....	24
2.2.3. IaaS.....	24
3.1. AMAZON WEB SERVICES.....	25
3.1.1. Definición.....	25
3.1.2. Servicios.....	28
3.2. TECNOLOGÍAS Y HERRAMIENTAS.....	30
3.2.1. Android.....	30
3.2.2. Bootstrap.....	31
3.2.3. Sirius – Ecore Modeling Framework.....	32
3.3. APLICACIONES EHEALTH.....	33
3.3.1. Características técnicas.....	33
CAPÍTULO 3: DESARROLLO DE LA ARQUITECTURA.....	35
3.1. DISEÑO DE LA ARQUITECTURA.....	35
3.1.1. Definición de requisitos y creación del metamodelo.....	35
3.1.2. Creación del metamodelo DSL visual.....	41
CAPÍTULO 4: IMPLEMENTACIÓN DE LA ARQUITECTURA EN EHEALTH.....	56
4.1. IMPLEMENTACIÓN.....	56
4.1.1. Aplicación móvil.....	59
4.1.2. Aplicación web.....	74
CAPÍTULO 5: PRUEBAS Y RESULTADOS.....	86
5.1. PRUEBAS DE ARQUITECTURA.....	86
5.2. PRUEBAS DE RENDIMIENTO.....	88

5.3. PRUEBAS DE FUNCIONAMIENTO.....	92
CAPÍTULO 6: CONCLUSIONES Y RECOMENDACIONES	95
5.1. CONCLUSIONES	95
5.2. RECOMENDACIONES.....	99
5.3. TRABAJOS FUTUROS.....	99
BIBLIOGRAFÍA.....	101

ÍNDICE DE FIGURAS

Figura 1	Universo MDE en el desarrollo de software dirigido por modelos	15
Figura 2	Arquitectura de software generativa que representa MDA	17
Figura 3	Componentes de un DSL	20
Figura 4	Metamodelo fundamental de Ecore	21
Figura 5	Arquitectura de cuatro niveles	22
Figura 6	Servicios populares de AWS	29
Figura 7	Paleta en EMF	39
Figura 8	Metamodelo que representa la arquitectura cross-device para eHealth	40
Figura 9	Estructura del proyecto	41
Figura 10	Estructura en forma de árbol	42
Figura 11	Metamodelo agregado a la representación diagrama	43
Figura 12	Viewpoint Specification Project	44
Figura 13	Propiedades de un contenedor	45
Figura 14	Contenedor Usuario y sus elementos	46
Figura 15	Propiedades de Conditional Style	46
Figura 16	Contenedor Proveedor y sus elementos	47
Figura 17	Configuración de propiedades de enlaces	48
Figura 18	Secciones para la paleta DSL	49
Figura 19	Propiedades de nodos y contenedores	50
Figura 20	Configuración de acciones de un nodo	50
Figura 21	Propiedades de los enlaces	51

Figura 22	Configuración de acciones de un enlace	51
Figura 23	Metamodelo representado con DSL y metamodelo en sintaxis abstracta	52
Figura 24	Creación de una representación viewpoint	54
Figura 25	Ejemplo de diseño una aplicación con el DSL	55
Figura 26	DSL del caso de uso con la aplicación eHealth with AWS	56
Figura 27	Consola de administrador en AWS	59
Figura 28	Proyecto eHealth en AWS	60
Figura 29	Servicios habilitados en el proyecto de la aplicación móvil	60
Figura 30	Autenticación por correo electrónico	61
Figura 31	Requisitos para el inicio y cierre de sesión del usuario	61
Figura 32	Habilitación de una tabla NoSQL de DynamoDB	62
Figura 33	Atributos de la tabla NoSQL del proyecto	62
Figura 34	Configuración de la nube para la aplicación en Android	63
Figura 35	Clases de Java y carpeta de recursos	63
Figura 36	Diagrama de secuencia de actividades del proyecto en Android Studio	64
Figura 37	Archivo JSON	65
Figura 38	Interfaz gráfica de la autenticación con AWS	67
Figura 39	Interfaz gráfica de la autenticación con AWS	67
Figura 40	Menú principal	68
Figura 41	Interfaz gráfica del podómetro	70
Figura 42	Interfaz gráfica del ritmo cardíaco	72
Figura 43	Actividad Tips OMS e Información	74
Figura 44	Hosting web y la URL pública	75

Figura 45	Página web alojada mediante Amazon S3 en una URL pública.....	76
Figura 46	Resumen de características del grupo de usuarios para la página web.....	77
Figura 47	Rol de IAM para la función de invocación de la API.....	78
Figura 48	Política de acceso de la función a DynamoDB	79
Figura 49	Rol de IAM con la política de acceso a DynamoDB	79
Figura 50	Política de las acciones permitidas para DynamoDB.....	80
Figura 51	Función de invocación con seguridad CORS.....	81
Figura 52	Testeo de la función de invocación exitosa.....	81
Figura 53	API de invocación de la función lambda	82
Figura 54	Método GET y el resumen de configuración y flujo de ejecución	82
Figura 55	Test de la API que invoca la función lambda	83
Figura 56	Obtención de la URL de invocación de la API	83
Figura 57	Consulta exitosa, a través del dominio correcto para CORS	85
Figura 58	Acceso denegado, a través del dominio incorrecto para CORS.....	86
Figura 59	DSL de escenario de prueba de arquitectura.....	86
Figura 60	Prueba de arquitectura con Web hosting AWS.....	87
Figura 61	Prueba de arquitectura con Web hosting GoDaddy	87
Figura 62	Prueba de carga con 500 usuarios concurrentes durante 60 segundos.....	88
Figura 63	Prueba de carga con 1000 usuarios concurrentes durante 60 segundos.....	89
Figura 64	Prueba de carga con 1500 usuarios concurrentes durante 60 segundos.....	90
Figura 65	Peticiones HTTP POST a Amazon Cognito	91
Figura 66	Peticiones HTTP POST a Amazon DynamoDB	91
Figura 67	Peticiones HTTP POST a Amazon DynamoDB	92

Figura 68 Verificación frecuencia cardíaca (a).....	93
Figura 69 Verificación frecuencia cardíaca (b).....	93
Figura 70 Verificación frecuencia cardíaca (c).....	94

ÍNDICE DE TABLAS

Tabla 1	<i>Preguntas de Investigación para el Mapeo Sistemático de la Literatura</i>	8
Tabla 2	<i>Preguntas de Investigación para el Revisión Sistemático de la Literatura</i>	9
Tabla 3	<i>Resumen de respuestas a preguntas de investigación</i>	12
Tabla 4	<i>SDKs disponibles en la plataforma AWS para el cliente</i>	27
Tabla 5	<i>Arquitectura de cuatro niveles para el Metamodelo de eHealth</i>	37
Tabla 6	<i>Atributos, Composición y Relaciones de las Clases del Metamodelo eHealth</i> .	38
Tabla 7	<i>Relación de correspondencia entre DSL y metamodelo abstracto</i>	53
Tabla 8	<i>SDK de sensores en dispositivo Samsung</i>	58
Tabla 9	<i>Servicios AWS utilizados en el sistema</i>	58

RESUMEN

El Internet en la actualidad es un pilar fundamental en diversos ámbitos de la vida cotidiana de los seres humanos. Cada día el número de personas que posee dispositivos con acceso a internet es mayor y las aplicaciones que se desarrollan buscan unirse al mundo del cloud computing, esta relación entre dispositivos y “cosas” en el internet es lo que se denomina Internet de las Cosas. Pero su gran complejidad de integrar dispositivos heterogéneos genera conflictos al momento de desarrollar aplicaciones específicas. De esta manera en el presente proyecto de titulación se presenta una arquitectura multiplataforma basada en ingeniería de modelos para generar aplicaciones en el dominio de eHealth. Para el diseño de la arquitectura se empleó técnicas de Model-Driven Development Environment (MDE) y Model-Driven Architecture (MDA), y se implementó un caso de uso específico que utilizó los servicios de Amazon Web Services (AWS). Finalmente se realizaron pruebas donde se evidenció que la arquitectura diseñada es multiplataforma y mediante el DSL se pueden construir más aplicaciones. En las pruebas de rendimiento de los servicios del caso de uso se demostró que el servidor AWS permite el escalamiento del sistema sin problemas y en cuanto a las pruebas de funcionamiento se verificó la ejecución correcta del sistema implementado.

- **ARQUITECTURA BASADA EN MODELOS (MDA)**
- **LENGUAJE DE DOMINIO ESPECÍFICO (DSL)**
- **COMPUTACIÓN EN LA NUBE**
- **INTERNET DE LAS COSAS**
- **EHEALTH**

ABSTRACT

The Internet today is a fundamental pillar in various areas of the daily life of human beings. Every day the number of people who have devices with Internet access is greater and the applications that are developed seek to join the world of cloud computing, this relationship between devices and "things" on the Internet is what is called the Internet of Things. But its great complexity of integrating heterogeneous devices generates conflicts when developing specific applications. In this way in the present project presents an architecture multiplatform based in engineering of models to generate applications in the domain of eHealth. For the design of the architecture, Model-Driven Development Environment (MDE) and Model-Driven Architecture (MDA) techniques were used, and a specific use case was implemented using Amazon Web Services (AWS). Finally tests were carried out where it was evidenced that the designed architecture is multiplatform and through the DSL more applications can be built. In the performance tests of the services of the use case it was demonstrated that the AWS server allows the escalation of the system without problems and as for the functional tests the correct execution of the implemented system was verified.

- **MODEL-DRIVEN ARCHITECTURE (MDA)**
- **SPECIFIC DOMAIN LANGUAGE (DSL)**
- **CLOUD COMPUTING**
- **INTERNET OF THINGS**
- **EHEALTH**

CAPÍTULO 1: DESCRIPCIÓN DEL PROYECTO

1.1. ANTECEDENTES

En el desarrollo histórico y la evolución de la tecnología es inminente el gran aporte que ha tenido el Internet y se considera que su desarrollo ha sido paralelo a la invención de dispositivos inteligentes tales como smartphones, tablets, computadoras personales entre muchos otros. Con ello se estima en (Espiniza, Pérez, & Peralta, 2017) que existen tres etapas antes de llegar a lo que hoy en día se conoce como Internet de las Cosas.

La primera etapa de esta evolución se la identifica como una interacción entre personas pues su objetivo era el intercambio de datos de texto y voz, por ejemplo con el uso de mensajería, correo electrónico y llamadas. En la segunda etapa se identifica ya una interacción entre máquina y persona pues en esta no era necesariamente la comunicación directa de personas sino más bien se centraba en compartir datos de videos, fotografías y otros contenidos mediante el Internet. Y finalmente la tercera etapa

denominada la interacción de máquina con máquina cuyo objetivo es automatizar la comunicación entre dispositivos y realizar tareas con la menor intervención humana.

Actualmente se hace cada vez más recurrente el término IoT o Internet de las Cosas, que no es nada más que el concepto que se le da a la conexión del mundo físico (dispositivos) mediante el Internet, todo ello con el objetivo de poder utilizar los datos de sus sensores, comunicarse entre ellos y así optimizar productividad y eficiencia de múltiples tareas, entrando al mundo del Cloud Computing.

Y finalmente en el 2010 el número de dispositivos conectados a Internet empezó a exceder el número de personas, llegando a una cifra aproximada de 1.84 dispositivos por persona (Evans, 2011), desde entonces el IoT ha empezado a generar un gran impacto en diversas áreas de aplicación, destacando cuatro dominios, el IoT personal y de Healthcare a escala de un individuo u hogar, el IoT empresarial a escala de una comunidad, el IoT de servicios públicos a escala nacional o regional y el IoT móvil que suele distribuirse en otros dominios principalmente debido a la naturaleza de la conectividad y la escala. (Gubbi, Buyya , Marusic, & Palaniswami, 2013).

La integración de IoT con cloud computing permite compensar algunas de las restricciones tecnológicas de los dispositivos que componen los sistemas IoT pues el cloud computing ofrece recursos, virtualmente ilimitados al Internet de las Cosas. (González García, 2017)

1.2 JUSTIFICACIÓN E IMPORTANCIA

La existencia de una variedad de sensores más económicos, la reducción del coste del ancho de banda y reducción del costo del procesamiento, la masificación del acceso a internet y el Big Data hoy en día permite mayor almacenamiento y procesamiento de información en la nube, todo esto en conjunto con la evolución de los smartphones, se están convirtiendo en la puerta de entrada a los dispositivos IoT actuando como un portal de control y gestión para los hogares, autos o dispositivos de salud y fitness que cada vez se están implementando más en el mercado (Morales).

Actualmente ya no se puede pensar en Deporte y Salud sin hablar de Tecnología (Romero, 2017). En el país y a nivel mundial hay una tendencia tecnológica hacia el área del deporte, salud y fitness, y hoy en día tal y como se menciona en (Rivas Costa, Anido Rifón, & Fernández Iglesias, 2016) se investigan y desarrollan ampliamente sistemas en los cuales es importante elementos como la consulta remota, el monitoreo de signos vitales y el monitoreo de actividades diarias tales como el ejercicio. Es por ello que se ha elegido ese campo de aplicación como una perspectiva a futuro para que se llegue a masificar en el país.

Sin embargo la diversidad de aplicaciones de IoT, ha generado la necesidad de disponer procesos de construcción guiados, debido a que cada una de las plataformas cuentan con sus propias tecnologías, lo que dificulta el desarrollo de nuevas aplicaciones, al tener que el diseñador, conocer todas estas.

MDA (Model-Driven Architecture), ofrece una metodología que enmarca el proceso de diseño en un patrón dado por un metamodelo, el cual dispone de diferentes niveles de abstracción, permitiendo que el diseño se enfoque en niveles específicos, desde lo más abstracto hasta la implementación. Y la OMG (Object Management Group) propone que el diseño se base en diagramas UML, permitiendo modelar los dominios específicos, tanto en lo referente al hardware como el software, a partir de herramientas como EMF (Eclipse Modeling Framework).

La aplicación eHealth with AWS que se realizará como caso de uso utilizará tecnologías de cloud de Amazon Web Services, aprovechando así de forma óptima los recursos del cliente y del proveedor al no tener que implementar costosas infraestructuras brindando mayor facilidad de mantenimiento al sistema y abaratando los costos de operación (Campoverde, Mazón , & Hernández, 2015). Además con el empleo de una arquitectura orientada a servicios (AOS) el servicio será consumido por cualquier aplicación que conecte a la URI (identificador uniforme de recurso) con lo que se amplía el ámbito de uso del sistema.

Al aportar con un metamodelo que permite la interoperabilidad de los sistemas heterogéneos, el cual es una de las características más importantes del IoT, y gracias a la fusión entre el cloud computing y el IoT, es posible llevar a cabo la interoperabilidad que se requiere en este tipo de sistemas, para brindar una mejor experiencia del usuario.

Además el ser multiplataforma destaca ventajas como aumentar la competencia y la integración de tecnologías existentes.

1.3. ALCANCE

Se propone un sistema de cloud computing en el dominio IoT de Healthcare fundamentado en una arquitectura de servicios, para alcanzar interoperabilidad entre múltiples plataformas tanto de hardware como de software. El sistema permitirá la gestión de sensores de actividad física dispuestos del lado del usuario, por medio de una aplicación móvil, y corresponderá a una instancia del metamodelo propuesto para dominios de IoT, basado en MDA.

El sistema planteado pretende acortar la brecha entre los dispositivos y el Internet, pues el principal inconveniente hoy en día del IoT es la interconectividad e interoperabilidad de los dispositivos con las diversas plataformas existentes, aportando con este proyecto a las personas que deseen monitorear sus signos vitales y mediante el monitoreo remoto sus médicos o entrenadores puedan establecer patrones y rutinas físicas, en base al ritmo cardíaco.

Al emplear ingeniería de modelos en el proceso de diseño del sistema, se separa cada una de las etapas, permitiendo determinar las características que tendrá el sistema. De esta manera el proyecto se enfocará en Amazon Web Services como alternativa de

proveedor de servicios Cloud, al ofertar la plataforma y el software como servicio para el despliegue del sistema.

En AWS se desarrollarán los servicios que deben usar tecnologías REST, la cual es utilizada por algunos dispositivos de hardware así como las plataformas móvil y web en el mundo de IoT. Finalmente, los sensores a emplearse utilizarán al teléfono móvil como portal de acceso al sistema en razón de tener por parte del usuario un medio para la gestión de su información.

La arquitectura que se propondrá permitirá que sea extendible a otras implementaciones del dominio de Healthcare, ya que se modelan distintos niveles de abstracción, que servirán de guía en el proceso de diseño, permitiendo que las capas intermedias se centren en los dispositivos, protocolos y plataformas que se emplean en IoT.

1.4. OBJETIVOS

1.4.1. Objetivo general

- Diseñar e implementar una arquitectura Cross-Device, basada en ingeniería de modelos para aplicaciones cloud, en ámbitos del eHealth.

1.4.2. Objetivos específicos

- Investigar el estado del arte de sistemas cloud de eHealth, la infraestructura como servicio, la plataforma como servicio y el software como servicio.
- Investigar las arquitecturas basadas en servicios y microservicios.
- Determinar los requerimientos de hardware y software para el sistema.
- Definir una arquitectura y un meta-modelo.
- Diseñar el sistema sobre la plataforma IoT seleccionada, para la comunicación con los dispositivos.
- Desarrollar la aplicación cliente que permita monitorear y dar sugerencias remotamente de la persona empleando tecnologías híbridas.
- Analizar los resultados.

1.5. ESTADO DEL ARTE

Para definir el estado del arte se utilizó la metodología de Mapeo Sistemático de la Literatura (SMS) y Revisión Sistemática de la Literatura (SLR), (Moguel Márquez, 2018), con el propósito de determinar los ámbitos más desarrollados en cuanto al tema de la investigación y encontrar aspectos que requieran mayor estudio y determinar aspectos de desarrollo que aporte ésta investigación a futuras investigaciones.

Tabla 1*Preguntas de Investigación para el Mapeo Sistemático de la Literatura*

Preguntas de investigación	Motivación
SMSP1: ¿eHealth es un dominio que acude a la tecnología de IoT y cloud computing actualmente?	Conocer el impacto que ha tenido la tecnología de IoT en el dominio de aplicaciones eHealth.
SMSP2: ¿Qué problemas de integración se han tenido en sistemas orientados a al cloud computing?	Conocer los campos de mayor dificultad para desarrollar un sistema de IoT y cloud computing.
SMSP3: ¿Qué arquitecturas definidas existen para la creación de sistemas eHealth?	Conocer la existencia de las arquitecturas propuestas para aplicaciones en el dominio de eHealth.

Para la metodología SMS para definir el estado del arte, se definieron preguntas de investigación claves, Tabla 1, con la intención de filtrar los resultados de artículos de revista, libros y publicaciones, que se enfoquen en la temática de investigación del presente proyecto. En SMS la búsqueda no es completamente técnica pero si describe el panorama actual en el que se encuentra el campo de investigación que se está desarrollando.

Para la metodología RSL, de la misma manera se definieron preguntas de investigación claves, Tabla 2, sin embargo en este caso difiere de SMS pues RSL se lo realiza con la motivación de conocer aspectos más técnicos en cuanto a la temática de investigación.

Tabla 2*Preguntas de Investigación para el Revisión Sistemático de la Literatura*

Preguntas de investigación	Motivación
SLRP1: ¿Qué características técnicas requieren mayor atención al momento de desarrollar un sistema de cloud computing e IoT?	Indagar sobre características, técnicas, tecnologías y protocolos que requieren mayor atención en los sistemas eHealth.
SLRP2: ¿Los sistemas de cloud computing eHealth se enfocan en los niveles de seguridad de la información implementando técnicas de encriptación o protocolos de seguridad?	Conocer el nivel de seguridad de los datos que se le debe dar a los sistemas cloud computing en el dominio de eHealth
SLRP3: ¿Qué tecnologías se implementan en el desarrollo de sistemas y arquitecturas de eHealth?	Conocer las tecnologías y arquitecturas que se han desarrollado para los sistemas eHealth.

Con las preguntas de investigación planteadas, se realiza la búsqueda de la información en fuentes oficiales de artículos, tesis o publicaciones, en este caso hicimos uso del buscador Google Scholar, ya que filtra las publicaciones y entrega fuentes confiables. Entre las investigaciones destacadas y que responden a las preguntas de SMS y SLR, se describen las siguientes:

En el dominio IoT de Healthcare, en el año 2013 el estudio de A. Kulkarni y S. Sathe (Kulkarni & Sathe, 2014), aporta con un trabajo que logra explicar varios casos de

usos de las aplicaciones de IoT en el ámbito de atención médica personalizada para lograr una atención excelente en costos accesibles, en este estudio se concluye que la intervención de IoT en los casos de Healthcare será cada vez mayor mientras la capa de servicio se establezca completamente y complete la infraestructura requerida en estas aplicaciones.

En el año 2015, Pedro Castillejo (Castillejo Parrilla, 2015) realiza aportaciones importantes en el ámbito IoT tales como solucionar problemas de heterogeneidad de los dispositivos y propone un mecanismo de seguridad para los datos médicos y biométricos de un usuario, contribuyendo y validando su trabajo con un caso de uso que brinda una solución para el control de parámetros físicos en entornos deportivos.

En el artículo de Prosanta Gope y Tzonelih Hwang (Gope & Hwang, 2015) el estudio de sus autores se centra en la privacidad de los datos de un paciente que es monitoreado mediante sensores corporales denominados BSN, destacando la importancia de la seguridad de los datos cuando se emplea IoT en el ámbito de la salud.

En el mismo año 2015, en el artículo de (Catarinucci, y otros, 2015) se propone una arquitectura específica de IoT para el monitoreo de pacientes con el uso de dispositivos biomédicos en hospitales y enfermerías, en donde hacen uso de tecnologías diversas y las integran mediante IPv6 sobre REST, obteniendo un sistema que brinda seguimiento de pacientes tanto localmente como remotamente.

En (Garai, Attila, & Péntek, 2016), los autores determinan la importancia de la protección de la información de salud del usuario en aplicaciones de este campo y su objetivo fue el planteamiento de una arquitectura que contemple la interoperabilidad de varios sistemas que conforman la Telemedicina, destacando el elemento seguridad pues en el campo de estudio va en auge y el intercambio de datos es cada vez mayor.

En la misma área de estudio de Healthcare, en el año 2017, Carmen Palao (Palao Cruz, 2017), diseña e implementa un sistema de comunicaciones que permite a pacientes con problemas cardiovasculares relacionados con arritmias poder llevar a cabo una monitorización constante de su ritmo cardíaco y detectar anomalías, aportando con un caso de uso más específico en el ámbito de la salud.

En el estudio desarrollado por Francisco Ortíz (Ortíz Bonilla, 2017), se desarrolla un sistema que realiza el monitoreo en tiempo real de pulso cardíaco con sensores Bluetooth LE de varios individuos a la vez y que pueden ser consultados en una aplicación móvil los datos que se almacenan en la nube de AWS, sistema que se piensa como una potencial implementación para gimnasios.

De acuerdo a los trabajos mencionados y contestando a las preguntas realizadas se elaboró la Tabla 3 de resumen en la cual se han clasificado las investigaciones y las preguntas a las que dan respuesta.

Tabla 3*Resumen de respuestas a preguntas de investigación*

Pregunta	Investigación	Respuesta
SMSP1	(Kulkarni & Sathe, 2014)	Los sistemas de health combinados con cloud computing e IoT cada vez tienen mayor demanda debido a sus beneficios y automatización de procesos.
	(Catarinucci, y otros, 2015)	
	(Palao Cruz, 2017)	
	(Ortíz Bonilla, 2017)	
SMSP2	(Castillejo Parrilla, 2015),	El problema mayor es la integración de sistemas heterogéneos.
SMSP3	(Castillejo Parrilla, 2015)	Arquitecturas propuestas se enfocan en la interoperabilidad de sistemas.
	(Garai, Attila, & Péntek, 2016)	
SLRP1	(Catarinucci, y otros, 2015)	Los tiempos de respuesta de los servidores, y principalmente la seguridad en la transferencia de datos.
	(Ortíz Bonilla, 2017)	
SLRP2	(Castillejo Parrilla, 2015)	Se han propuesto diferentes mecanismos para cumplir los estándares de seguridad, mediante autenticación y encriptación.
	(Gope & Hwang, 2015)	
	(Garai, Attila, & Péntek, 2016)	
SLRP3	(Castillejo Parrilla, 2015)	Sistemas que funcionan mediante la red ya sea localmente o remotamente, y en cuanto a sensores utilizan tecnología Bluetooth LE.
	(Catarinucci, y otros, 2015)	
	(Garai, Attila, & Péntek, 2016)	
	(Ortíz Bonilla, 2017)	

Como se ha podido destacar existe una variedad de investigadores estudian nuevas aplicaciones y arquitecturas para proporcionar un marco de desarrollo que

estandarice las diferentes tecnologías e impulse más aun el despliegue de IoT en el área de Healthcare. De tal manera que en este proyecto se desea proponer en primera instancia un metamodelo que defina una arquitectura genérica para una familia de aplicaciones IoT en el ámbito de Healthcare, y cuya aplicación específica será el desarrollo un sistema eHealth que utilice Web Services y permita la interoperabilidad de plataformas móvil y web.

CAPÍTULO 2: MARCO CONCEPTUAL

2.1. DESARROLLO DE SOFTWARE DIRIGIDO POR MODELOS

Para hablar del desarrollo de software dirigido por modelos es importante comprender la relación entre conceptos como modelo y metamodelo; un modelo representa aspectos de interés para un sistema en este caso de software y ésta representación se expresa en lenguajes de modelado, de ahí se deduce que el metamodelo es el conjunto de modelos, que será una definición abstracta del problema especificado (García, y otros, 2012).

El desarrollo de software dirigido por modelos es la evolución de la ingeniería del software en un esfuerzo por mejorar y automatizar los procesos y la calidad de las aplicaciones que se crean para que tengan un nivel de abstracción mucho mayor, y esto lleva a los desarrolladores a acortar tiempos en elaboración de código.

Es así que surge MDE (Model-Driven Development Environment) que es una rama de la ingeniería de software que se dedica al uso metódico de modelos de software, de tal manera que mejore la productividad, el mantenimiento y la interoperabilidad de sistemas. (García, y otros, 2012)

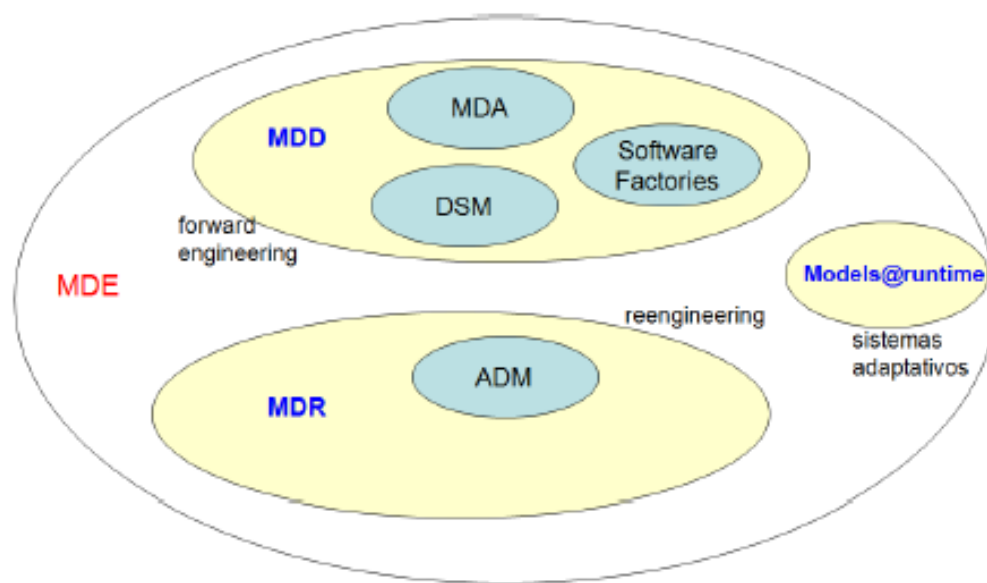


Figura 1 Universo MDE en el desarrollo de software dirigido por modelos

Fuente: (García, y otros, 2012)

Tal y como se representa en la Figura 1, MDE es un universo de paradigmas del desarrollo del software cuyos procesos conducen a la creación de aplicaciones mediante Model-Driven Development (MDD), a la deducción de las mismas a través de Model-Driven Runtime (MDR), y a los sistemas adaptativos.

En el universo de modelado MDD se destaca MDA (Model-Driven Architecture), que es uno de los procesos de la ingeniería directa basada en modelos para la creación

de una aplicación. Todo este conjunto de procesos comparten características importantes, tales como se mencionan en (García, et al., 2012):

- Utilizan modelos que puede representar un sistema de manera total o parcial.
- Un modelo debe representarse mediante lenguajes de modelado llamados lenguajes específicos del dominio (DSL).
- El metamodelo representa un DSL.
- Al traducir los modelos a código mediante transformaciones de modelos se consigue la automatización de procesos.

Con estas características de todas las técnicas y herramientas de MDE se busca el cumplimiento de su objetivo principal que es la automatización de los procesos a la hora de desarrollar software de calidad optimizando tiempo.

2.1.1. MDA

La OMG (Object Management Group) especifica que MDA es una de sus aproximaciones basadas en modelos (Gaitán Peña, 2017) y de acuerdo con Stahl y Voelter (García, et al., 2012) se considera como una arquitectura de software generativa en general.

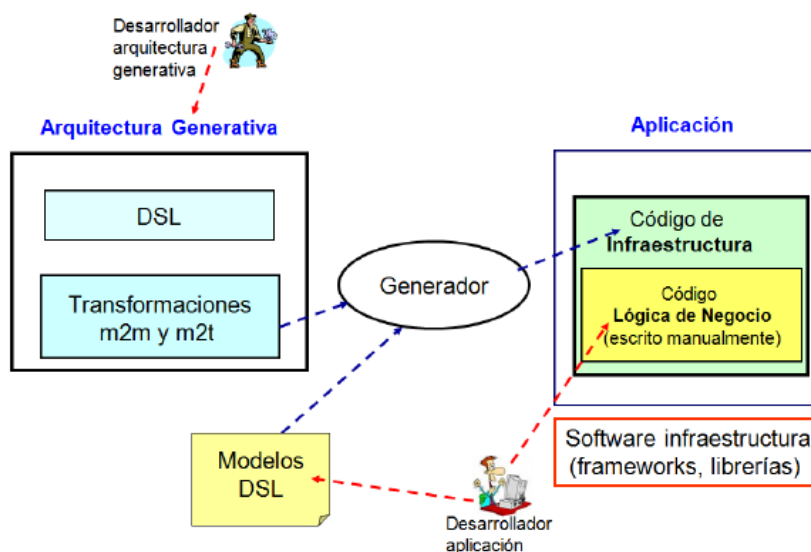


Figura 2 Arquitectura de software generativa que representa MDA

Fuente: (García, y otros, 2012)

De acuerdo a la Figura 2 el objetivo fundamental en MDA de una aplicación es aislar su funcionalidad de su implementación en una plataforma específica, de tal forma que se puedan crear modelos independientes de la plataforma (García, y otros, 2012), con lo que se conseguiría un modelo general y que se pueda generar una familia de aplicaciones, o bien se realicen transformaciones de modelos según se requiera.

Como MDA es una visión particular de MDE, ésta es ilustrada con lenguajes de modelado para la representación total o parcial de un sistema de software, con la visión de automatizar el proceso de traducción de modelo a código de infraestructura. (García, et al., 2012)

Las ventajas destacadas de MDA son: independencia de plataforma, interoperabilidad entre plataformas y por ende la reutilización de aplicaciones.

2.1.2. Niveles de abstracción

Los modelos recomendados por MDA (García, et al., 2012) son los denominados niveles de abstracción que pueden ser utilizados en los diversos sistemas que se quieran desarrollar, los niveles básicos que se definen son: CIM (Computation Independent Model), PIM (Platform Independent Model) y PSM (Platform Specific Model).

La OMG detalla diversos niveles y como se relacionan, pero no describe concretamente como crear estos niveles de abstracción, su representación y transformaciones (Kardoš & Drozdová, 2010). Sin embargo, con la importancia que poco a poco va obteniendo MDA se han ido generando recomendaciones de diferentes investigadores.

El nivel CIM se enfoca en el sistema como negocio (Gaitán Peña, 2017), y es aquel que detalla las actividades y funciones del sistema sin mostrar detalles de su elaboración, este modelo detalla el contexto en el que funciona el sistema y explica que esperar del mismo. En el CIM no se dan características de modelos o dispositivos que se utilicen en la implementación del sistema, por ende es un nivel de abstracción que sirve como traductor entre los expertos del negocio y los expertos del software.

PIM es el nivel de abstracción en el que se describen las características o funcionalidades del software (Gaitán Peña, 2017) que requiere el sistema pero sin

especificar nada referente a la tecnología que implemente, es decir, es independiente de la plataforma en que se pueda emplear el sistema. Este nivel se representa principalmente por modelos UML (Unified Modeling Language), para el propósito de MDA los diagramas UML más utilizados son: diagramas de casos de uso, diagramas de actividad, diagramas de secuencias y diagramas de dominios o modelos (Kardoš & Drozdová, 2010).

El modelo PSM es el que especifica el sistema con lo detallado en PIM mas las especificaciones de la plataforma que se utilizará en el mismo, por ende su lenguaje es manipulado por los desarrolladores de software y puede ser más técnico que los anteriores modelos dependiendo de su objetivo. En este nivel se genera el código (Gaitán Peña, 2017).

2.1.3. Lenguaje de modelado

El lenguaje de modelado o mejor conocido como lenguaje específico de dominio DSL que deriva de los UML, se utiliza para determinar y caracterizar un modelo, estos pueden ser gráficos o textuales y sus componentes clave según (García, y otros, 2012) son: la sintaxis y la semántica.

Estos dos aspectos del lenguaje de modelado son fundamentales, pues con la sintaxis se determina expresiones que definen el modelo y la semántica determina la simbología de las expresiones de sintaxis.

La sintaxis es de dos tipos concreta y abstracta, donde la primera se dedica a la notación gráfica para los diagramas y la segunda corresponde a la estructura lógica y textual que tendrán los diagramas. En cuanto a la semántica es de interpretación y derivación, donde la primera determina la correspondencia con la realidad del modelo y la segunda tiene que ver con la derivación que puede existir de otros modelos.

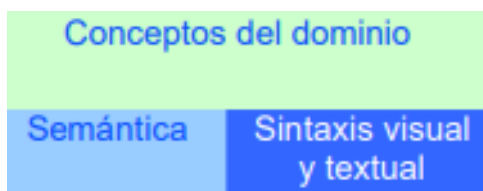


Figura 3 Componentes de un DSL

Fuente: (Mora, Ruiz, García, & Piattini, 2006)

Ahora bien, los lenguajes de metamodelado corresponden a la sintaxis abstracta del modelo, es decir, al metamodelo (García, y otros, 2012) y el lenguaje de metamodelado más popular en este ámbito es Ecore de Eclipse Modeling Framework (EMF).

Ecore se implementó por EMF porque es un subconjunto del lenguaje de metamodelado MOF (Meta Object Facility), pues la OMG considera que MOF que define UML es muy extenso y complicado (García, y otros, 2012); con Ecore se determina clases, atributos y referencias excepto asociaciones como lo hace MOF (Gaitán Peña, 2017).

Ecore se puede representar asimismo con su propio metamodelo, el cual se maneja con cuatro clases elementales (Steinberg, Budinsky, Merks, & Paternostro, 2008) que conforman el metamodelo fundamental de Ecore de la Figura 4:

- EClass: representa una clase metamodelada, posee nombre, atributos y referencias.
- EAttribute: representa un atributo metamodelado, poseen un nombre y un tipo.
- EReference: representa una relación de clases, posee un nombre y una bandera que indica si es agregación o referencia.
- EDataType: representa el tipo de un atributo, puede ser un tipo de dato primitivo o un objeto de Java.

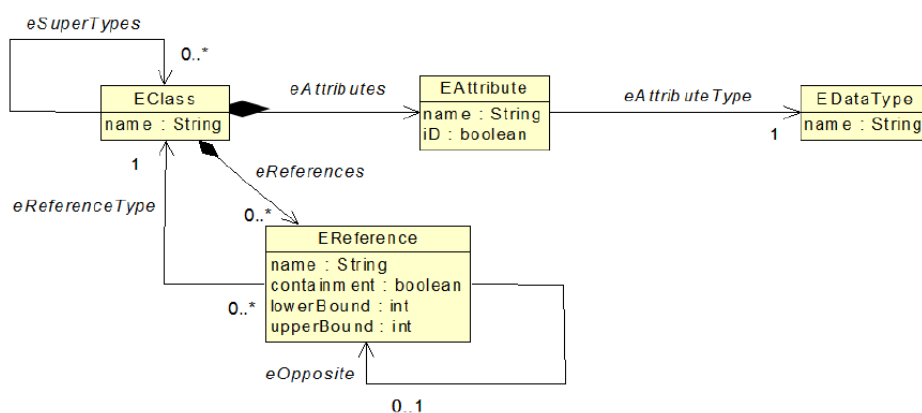


Figura 4 Metamodelo fundamental de Ecore

Fuente: (García, y otros, 2012)

Debido a la relación estrecha entre modelo y metamodelo, es fundamental comprender la arquitectura de cuatro niveles (García, y otros, 2012) en la cual se establece:

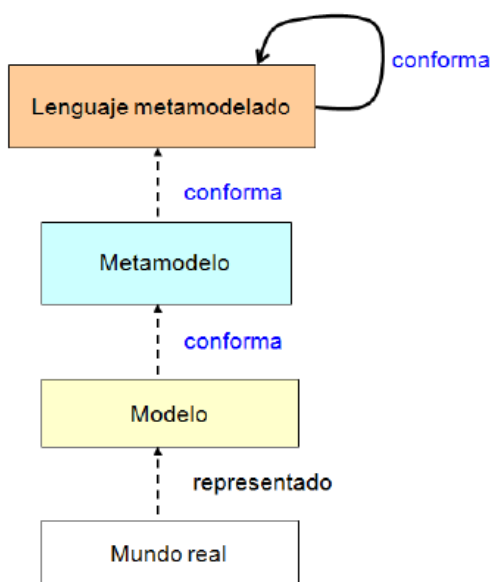


Figura 5 Arquitectura de cuatro niveles

Fuente: (García, y otros, 2012)

- M0→ Nivel de instancias de objetos: representa los conceptos del mundo real que tienen impacto en el software.
- M1→ Nivel de modelo: representa el modelo del nivel M0, es decir, el modelo del sistema.
- M2→ Nivel de metamodelo: representa el modelo del nivel M1, es decir, el metamodelo del sistema.
- M3→ Nivel de meta-metamodelo: representa el modelo del nivel M2, es decir, el meta-metamodelo del sistema.

En la Figura 5 se visualiza la arquitectura de cuatro niveles, donde el nivel M3 considera a los lenguajes de metamodelado como Ecore, por ello éste se define con sus propios conceptos con los que desarrolla otros metamodelos (García, y otros, 2012).

2.2. MODELOS DE SERVICIO CLOUD

Cloud Computing es el modelo que utiliza Internet como una nube, la cual suministra recursos y servicios a los usuarios; estos pueden ser infraestructura, plataforma y aplicaciones (Murazzo & Rodríguez, 2010). Sus capas son las siguientes:

2.2.1. SaaS

En el software como servicio o SaaS se provee al usuario de aplicaciones que requiera mediante el acceso a Internet (IBM, 2015), es decir, que los usuarios obtienen la aplicación del proveedor al cual pueden acceder a través de su navegador web o API para consumir sus servicios.

Entre sus características principales se tiene (IBM, 2015):

- La utilización de los recursos es escalable.
- Alta compatibilidad entre dispositivos y el acceso a las aplicaciones, siempre que tenga conexión a Internet.
- Seguridad de datos en caso de fallos del dispositivo.
- Las actualizaciones son automáticas para el usuario y su gestión corresponde a los proveedores SaaS.

2.2.2. PaaS

En la Plataforma como servicio o PaaS se provee al usuario de la Infraestructura pero sin tener acceso a la gestión de la misma, es decir, se suministra al usuario las herramientas necesarias para desarrollar las aplicaciones que desee (IBM, 2015).

Entre sus características principales se tiene (IBM, 2015):

- El usuario crea sus aplicaciones con las herramientas provistas del proveedor sin preocupación de la gestión de la infraestructura física.
- Trabajos colaborativos entre equipos remotos.
- La gestión de PaaS se centra en la seguridad de datos.

2.2.3. IaaS

En la Infraestructura como servicio o IaaS se provee a los desarrolladores la infraestructura necesaria para el almacenamiento de datos, las redes, los servidores y demás recursos de hardware (IBM, 2015) para permitir que la aplicación que se desee implementar pueda escalar sin necesidad de preocuparse por la infraestructura, pero si permite la gestión de la misma, contrario a PaaS. Este modelo de servicio generalmente es pagado de acuerdo al consumo y se lo utiliza para generar aplicaciones a gran escala.

Entre sus características principales se tiene (IBM, 2015):

- Acceso a la gestión de los recursos de hardware pero sin comprarlos.
- Escalamiento de la infraestructura de acuerdo a la demanda de almacenamiento y procesamiento.

3.1. AMAZON WEB SERVICES

Una de las soluciones imponente en la nube es el brindar acceso a servicios y recursos mediante Internet, en ese campo ya existen varios proveedores que compiten por ser los líderes, entre ellos se tiene Amazon Web Services, Microsoft Azure y Google Cloud (TechTarget, 2016) como principales aunque la lista sea extensa. Cada uno de estos proveedores tiene sus características y servicios que ofertan y es decisión del cliente escoger cual considera mejor opción. Pueden variar los factores entre precio, capacidad y velocidad de procesamiento.

3.1.1. Definición

Amazon Web Services (AWS) es una plataforma que cuenta con un conjunto amplio de servicios informáticos en la nube que provee a sus clientes una gran variedad de características y herramientas de desarrollo para crear las soluciones de sus aplicaciones, con la finalidad de escalar sin la preocupación de la infraestructura, por ello AWS pertenece al modelo de servicio cloud IaaS.

AWS es una de las plataformas de computación en la nube pioneras desde el 2006, y como se mencionó es la competencia directa de otras plataformas como Windows Azure y Google Cloud. Algunas de las características más importantes que posee AWS son:

- Seguridad: Cuenta con un respaldo considerable de certificaciones y auditorías reconocidas para brindarle al cliente la garantía de procesos y políticas implementadas, tales como HIPA, ISO 9001, ITAR, CSA, PCI DSS Nivel 1, ISO 27001, MTCS de nivel 3, CSM del DoD, niveles 1-2, 3-5, entre otras (AWS, Amazon Web Services, 2018).
- Infraestructura: La infraestructura de la plataforma se encuentra localizada en varias regiones a nivel mundial, permitiendo al cliente realizar sus implementaciones de acuerdo a su ubicación obteniendo una latencia mínima y mejor desempeño en su aplicación (AWS, Amazon Web Services, 2018).
- Almacenamiento: Varios servicios ofertados por la plataforma brindan al cliente opciones de almacenamiento con ventajas y características importantes como lo son un cifrado integrado que permite tener mayor seguridad en los datos, así como también un almacenamiento en las bases de datos que permita la escalabilidad según se requiera, logrando una latencia imperceptible en el número de lecturas y escrituras (AWS, Amazon Web Services, 2018).
- Escalamiento: El tener acceso a la infraestructura de acuerdo a la necesidad permite que el cliente configure la capacidad de los recursos que requiera para su aplicación sin que éste se deba preocupar por implementar hardware y centros de

datos, además el mantenimiento de la infraestructura es responsabilidad del proveedor, lo cual disminuye costos de administración y tiempo de desarrollo, aumentando productividad. (AWS, Amazon Web Services, 2018)

- Documentación y SDKs: La plataforma posee una extensa colección de documentación y tutoriales sobre todos los servicios que oferta, dando al cliente todas las herramientas posibles para un mejor entendimiento e implementación (AWS, Amazon Web Services, 2018).

Con la ayuda de las herramientas y SDKs que brinda AWS, el cliente puede construir y administrar de una manera más efectiva grandiosas aplicaciones en la nube, de acuerdo a sus necesidades y conocimientos de lenguajes de programación. En la Tabla 4 se presentan los SDKs que maneja la plataforma:

Tabla 4
SDKs disponibles en la plataforma AWS para el cliente

SDKs de lenguaje de programación	Java	Node.js	Python
	Go	.Net	C++
	JavaScript	Ruby	PHP
SDKs de dispositivos IoT	Embeddded C	JavaScript	Arduino Yun
SDKs de dispositivos móviles	Java	Python	C++
	Android	iOS	Xamarin
	Unity	React Native	

Fuente: (Mistry, 2018)

3.1.2. Servicios

AWS dispone de una cantidad considerable de servicios a para sus clientes, prácticamente para cualquier implementación en la nube que se requiera, como se observa en la Figura 6, donde se listan los servicios más comunes y utilizados que se encuentran categorizados para la implementación, administración, y aplicación deseada.

Se debe considerar que AWS permite al cliente crear una cuenta sin costo, al realizarlo la plataforma le concede una capa de servicios gratuita durante un año y otros servicios no vencen, en cuanto a los servicios pagados el costo va de acuerdo a la escala del proyecto y los recursos que consume (AWS, Capa gratuita de AWS, 2018).

Para el caso de uso de este proyecto se ha examinado las características y costos de los servicios fundamentales considerados para la aplicación (AWS, Capa gratuita de AWS, 2018), se define para que sirven y si aplica la capa gratuita que oferta el proveedor:

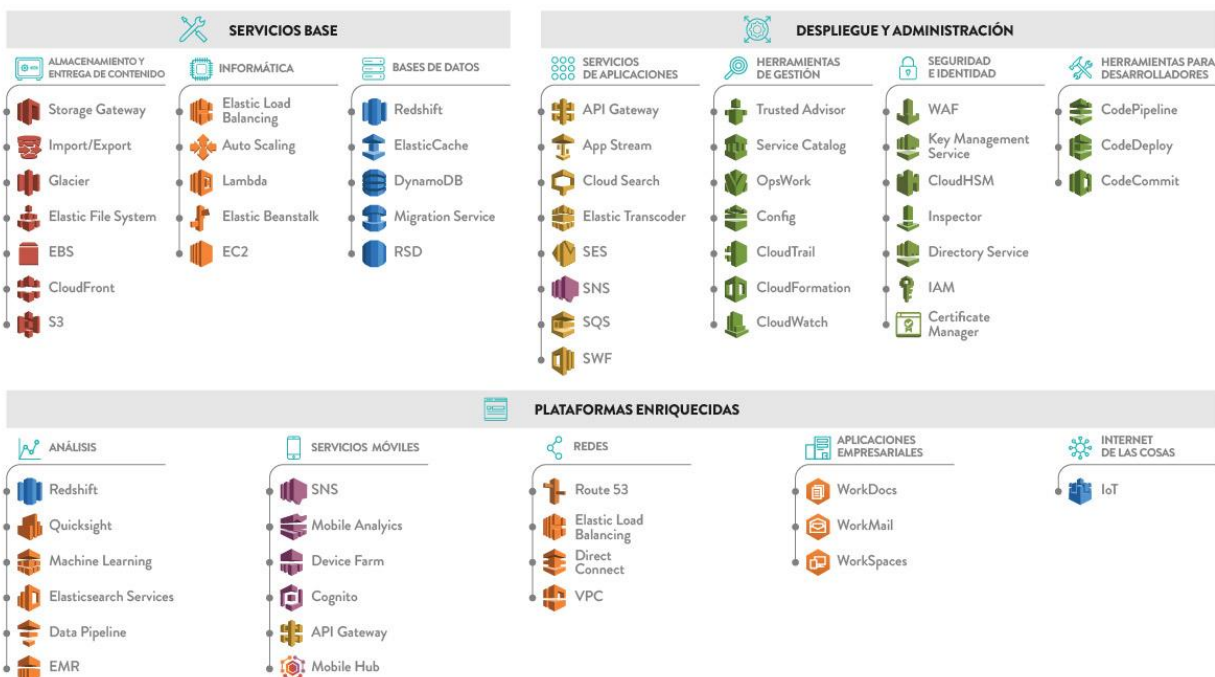


Figura 6 Servicios populares de AWS

Fuente: (GPC, 2017)

- Amazon Cognito para autenticar los usuarios del dispositivo móvil y de la aplicación web al backend. Es parte de la capa gratuita sin vencimiento, hasta 50000 MAU (usuarios mensuales activos) por mes.
- Amazon IAM (Identity and Access Management) para definir permisos a los usuarios tanto del dispositivo móvil, como de la aplicación web. Es parte de la capa gratuita sin vencimiento.
- Amazon DynamoDB como base de datos no relacional para almacenar información de la aplicación. Es parte de la capa gratuita sin vencimiento hasta 25GB de capacidad, más de eso se facturará un valor.

- Amazon S3 como contenedor de la página web y los archivos que requiere la misma. Es parte de la capa gratuita con un límite de hasta 5GB de almacenamiento.
- Amazon Lambda para la creación de funciones de invocación para extraer información desde DynamoDB. Es parte de la capa gratuita sin vencimiento, limitado para un millón de solicitudes al mes.
- Amazon API Gateway para la creación de la API RESTful que conecta al cliente web con la función Lambda que extrae los datos de DynamoDB. Es parte de la capa gratuita, limitado para un millón de llamadas de API (Application Programming Interface) mensuales.
- AWS Mobile Hub es la consola de Amazon que permite crear aplicaciones móviles con determinados servicios de la plataforma, para este caso la aplicación que convertirá al celular en el dispositivo IoT. No aplica un costo adicional.

3.2. TECNOLOGÍAS Y HERRAMIENTAS

3.2.1. Android

Android es un sistema operativo de código abierto basado en Linux creado para una gran diversidad de dispositivos móviles. El entorno de desarrollo integrado (IDE) oficial para el desarrollo de aplicaciones con este sistema operativo es Android Studio. Entre sus características principales se tienen (Android, 2018):

- El sistema de compilación se basa en Gradle.
- Gracias a un entorno unificado se puede desarrollar aplicaciones para cualquier dispositivo Android.
- Posee plantillas de diseño de las actividades y frameworks más comunes en las aplicaciones.
- Se puede vincular con tu cuenta de GitHub y se puede importar códigos de ejemplo.
- Posee Instant Run para aplicar cambios a la aplicación ejecutándose, sin tener que compilar nuevamente el APK.
- Cuenta con herramientas para detectar problemas de rendimiento, usabilidad, compatibilidad de versión, entre otros.
- Compatible con lenguajes como C++ y NDK.
- Soporte para Google Cloud Platform, facilitando la integración de Google Cloud Messaging y App Engine.
- Posee un emulador para realizar las pruebas de la aplicación.

3.2.2. Bootstrap

Bootstrap es un framework HTML, CSS y JavaScript para el diseño frontend de páginas web responsive que permite utilizar muchos elementos web tienen (Bootstrap, 2018) desde iconos a desplegables, obteniendo aplicaciones con gran calificación desde la perspectiva de experiencia del usuario, pues su diseño visual final suele ser muy

intuitivo y de fácil entendimiento para el cliente. Entre sus principales características se tienen:

- Simplifica en gran medida la maquetación por columnas con grid system para páginas web, resultando aplicar buenas prácticas y estándares de diseño.
- Organización visual más efectiva de una página web.
- Combina los lenguajes de HTML5, CSS y JavaScript para obtener una página web funcional, dinámica y buen contenido visual.
- Diseño responsive, es decir, diseño que se adapta a la pantalla de cualquier dispositivo sin importar su escala o resolución.
- Integración correcta con las principales librerías JavaScript, JQuery y Ajax.
- Su creador fue Twitter, por tanto garantiza muchos procesos con una comunidad de soporte activa.
- Se puede implementar externamente con WordPress, Drupal, etc.
- Permite el uso de Less, que enriquecen los estilos de la web.

3.2.3. Sirius – Ecore Modeling Framework

Sirius es uno de los proyectos de Eclipse (Eclipse, Sirius, 2018) cuya finalidad es ayudar a construir un conjunto de herramientas de modelado gráfico, se encuentra beneficiado por las tecnologías de modelado de Eclipse EMF y GMF.

Obeo y Thales lo crearon para brindar un soporte al campo de la ingeniería de arquitectura basada en modelos, encaminado a la adaptación de sistemas específicos. El beneficio más destacado de Sirius, (Eclipse, Sirius, 2018), es que hace posible aprovisionar sistemas que requieran batallar con arquitecturas complejas en dominios específicos.

3.3. APLICACIONES EHEALTH

El término eHealth es el que se le denomina al área de la asistencia sanitaria basada en Internet, dentro de IoT este es uno de los campos en donde se realizan muchas aplicaciones en las que se utilizan diversos protocolos y tecnologías para la comunicación. (Sebestyen, Hangan, Oniga, & Gál, 2014)

Los subcampos de eHealth consideran áreas de consultas médicas remotas, monitoreo en tiempo real, supervisión de pacientes y administración de actividades hospitalarias, eHealth puede llegar a convertirse en una gran ayuda para el Estado en países donde los recursos destinados a la Salud sean limitados. (Sebestyen, Hangan, Oniga, & Gál, 2014)

3.3.1. Características técnicas

Las aplicaciones de eHealth que se desarrollen tienen que tener en cuenta una serie de características técnicas, entre las que destaca la seguridad de los datos (Sebestyen, Hangan, Oniga, & Gál, 2014), en este sentido, todos los estudios de este

tema puntualizan la importancia de la implementación de protocolos de seguridad que debe garantizarles a los usuarios que sus datos son manejados con responsabilidad.

En los sistemas de monitoreo en tiempo real, se agrega otra característica técnica fundamental para su desarrollo, que es la velocidad de transmisión de los datos (Sebestyen, Hangan, Oniga, & Gál, 2014) y su correcta recepción en los servidores, sin alteraciones.

Otra característica técnica es considerar el tipo de datos a transmitir, y es en el caso de los dispositivos en los que existe tal variedad, que la unificación de un tipo de dato es realmente complejo. En el caso de aplicaciones estrictamente médicas existe por ejemplo un estándar DICOM para lo referente a transmisión de imágenes médicas, o HL7 para datos de ECG. (Sebestyen, Hangan, Oniga, & Gál, 2014)

La información útil que se transmite también es un punto a considerar en estos sistemas, pues debido a los encabezados que se les agregan a las tramas para transmitir la información al Internet, su velocidad puede disminuir y en casos de aplicaciones de medicina que sean estrictamente en tiempo real es crítico.

CAPÍTULO 3: DESARROLLO DE LA ARQUITECTURA

3.1. DISEÑO DE LA ARQUITECTURA

El objetivo de desarrollar la arquitectura es representar los conceptos y las relaciones del metamodelo para cloud computing que permita elaborar diferentes casos de uso en el dominio de eHealth, obteniendo la descripción de un modelo general que se pueda aplicar al desarrollo de una familia de aplicaciones en el ámbito mencionado.

Para este propósito se utilizó el software Ecore que es un lenguaje de modelado de Eclipse basado en MOF (Meta-Object Facility).

3.1.1. Definición de requisitos y creación del metamodelo

Primeramente se definieron las características y requisitos generales que se considera debe poseer una arquitectura para una familia de aplicaciones eHealth con cloud computing:

- Sistema multiusuario.
- El usuario puede tener uno o varios dispositivos.
- El usuario poseerá un dispositivo que tenga sensor o sensores y mediante una aplicación pueda conectarse a la nube, convirtiéndose así en el nodo IoT.
- Un dispositivo puede poseer uno o varios sensores.
- El caso de uso que se desarrolle se puede componer de uno o varios proveedores del sistema general.
- La aplicación puede consumir uno o todos los servicios que oferte el proveedor.
- Los servicios básicos del proveedor IaaS deben ser: base de datos, API Rest, funciones backend, autenticación y hosting.

En base a las características generales planteadas, para el desarrollo del metamodelo se trasladó aquellas ideas a la arquitectura de cuatro niveles descrita en el capítulo anterior, para ello se definió los conceptos que representen cada nivel.

De acuerdo a éste análisis se elaboró una tabla con los elementos que se compone el metamodelo y su respectiva descripción, como se muestra en la Tabla 5:

Tabla 5*Arquitectura de cuatro niveles para el Metamodelo de eHealth*

Nivel	Descripción	Elementos
M3	Metamodelo Ecore	EClass, EAttribute, Ereference, EDataType
M2	Metamodelo Arquitectura eHealth	Architecture, User, Provider, Device, Sensors, SensorType, ModelDevice, MobileApp, WebApp, Services, DataStore, LogicBackend, APIRest, Hosting, Authentication.
M1	Modelos del sistema real de Arquitectura eHealth	Arquitectura, Usuario, Proveedor, Servicio, Sensor, Tipo de sensor, Modelo de dispositivo, App móvil, App web, Base de datos, Función de backend, API Resfull, Hosting, Autenticación.
M0	Sistema real eHealth	Arquitectura raíz, Usuarios, Proveedores, Dispositivos y sus tipos, Sensores y sus tipos, Aplicación móvil, Aplicación Web, Servicios en la nube, Bases de Datos, Funciones, API's Restful, Hospedaje de páginas web, Autenticaciones.

Fuente: (García, y otros, 2012), adaptado por: Alava M.

A partir de la arquitectura de cuatro niveles aplicada al metamodelo que se desarrolló, se tiene una idea más concreta de las clases necesarias, sin embargo, también se debía identificar cada una de las clases con sus características, composiciones entre clases y sus relaciones.

Tabla 6*Atributos, Composición y Relaciones de las Clases del Metamodelo eHealth*

Clase	Atributos	Composición	Relaciones
Architecture	Nombre (String)	-	-
User	Nombre (String)	Architecture	-
Device	Modelo (String)	User	MobileApp WebApp
Sensors	Tipo (String)	Device	-
Provider	Nombre (String)	Architecture	-
MobileApp	Id (String)	Provider	Authentication
WebApp	Id (String)	Provider	Authentication
Services	Nombre (String)	Provider	-
Authentication	Nombre (String)	Services	APIRest LogicBackend
Hosting	Nombre (String)	Services	WebApp DataStore
APIRest	Nombre (String)	Services	Authentication LogicBackend
LogicBackend	Nombre (String)	Services	APIRest DataStore
DataStore	Nombre (String)	Services	LogicBackend Hosting

Para éste propósito, se elaboró la Tabla 6 con cada una de las clases identificadas y se definió los atributos que deberían poseer, sus relaciones y composiciones.

Una vez definida cada una de las clases para el metamodelo, se procedió a la construcción del mismo en el software Eclipse Modeling Framework. Se creó un proyecto del tipo Ecore Modeling Project, el cual brinda las herramientas necesarias para plasmar gráficamente el diagrama del metamodelo de acuerdo a lo definido en la Tabla 6.

Los elementos que se utilizaron de la paleta, fueron las clases, datatype, referencias, referencias bidireccionales y composición.

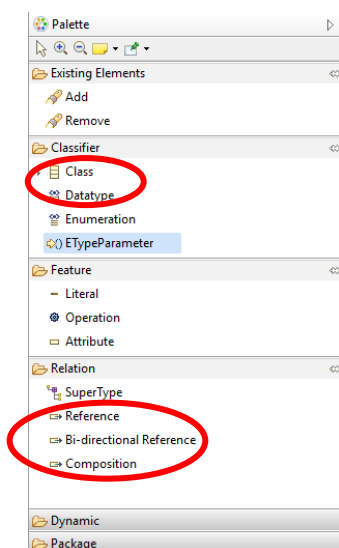


Figura 7 Paleta en EMF

El metamodelo final diseñado con todos los elementos y definiciones se representa en la Figura 8.

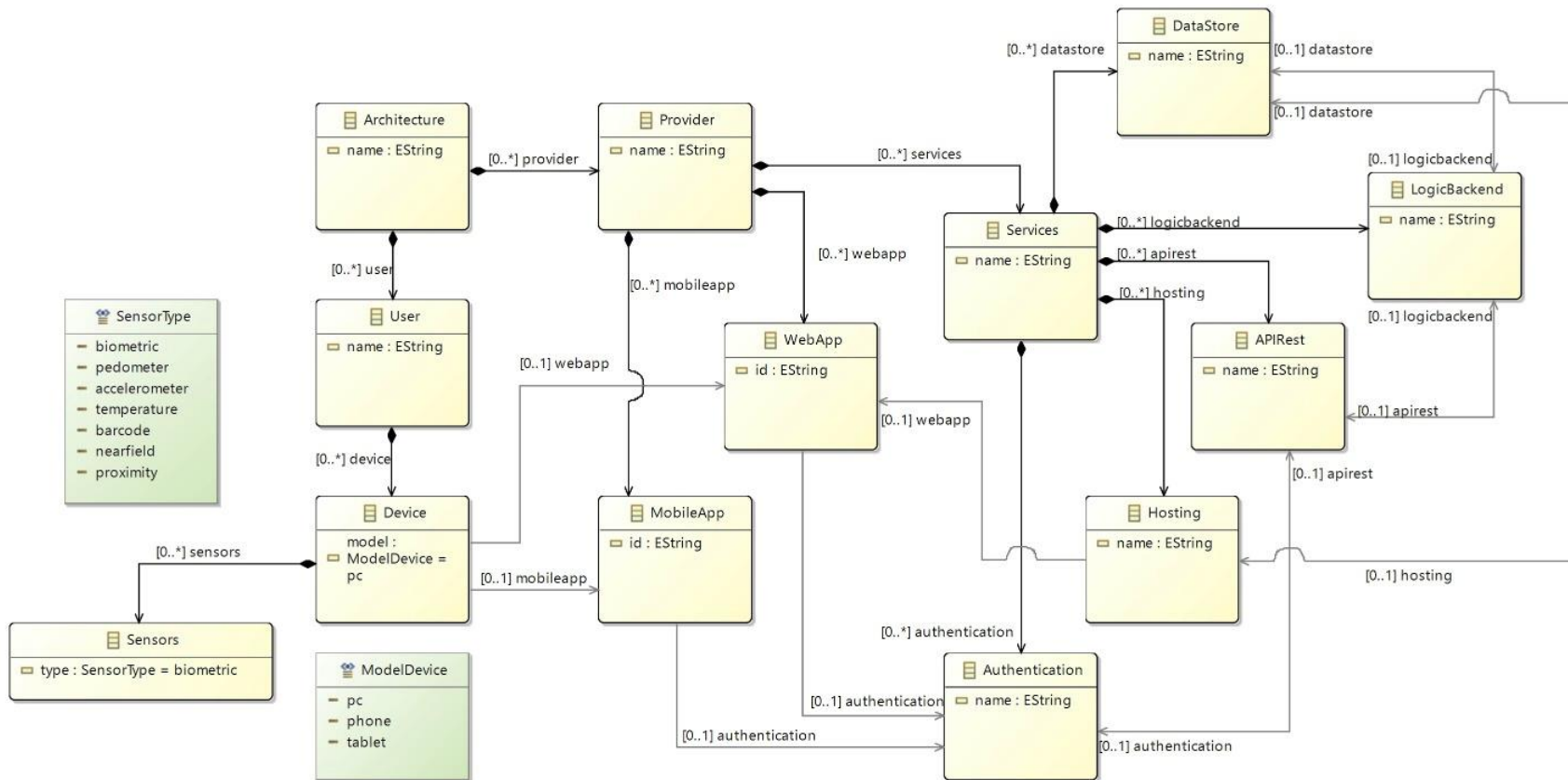


Figura 8 Metamodelo que representa la arquitectura cross-device para eHealth

3.1.2. Creación del metamodelo DSL visual

El metamodelo que se desarrolló en la Figura 8 es la arquitectura cross-device para cloud computing que se va a aplicar a un sistema ehealth, debido a que su interpretación puede resultar difícil debido a su naturaleza abstracta, se elaboró su DSL para una interacción más familiar con los desarrolladores.

Cuando se creó el metamodelo se generaron una serie de archivos que se deben tener en cuenta para el siguiente paso que es la creación del DSL. Específicamente con el archivo .genmodel se ejecutó el nuevo entorno de Eclipse.

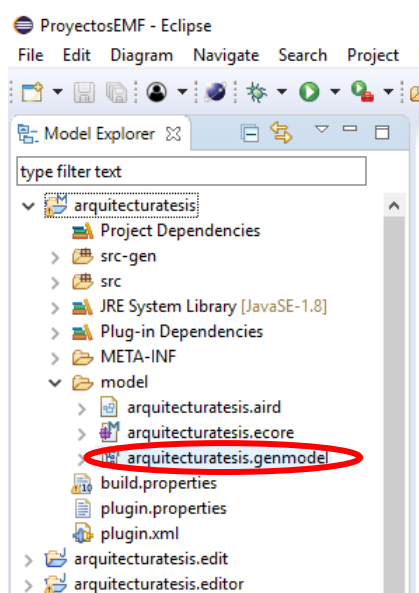


Figura 9 Estructura del proyecto

Al abrir el nuevo entorno de Eclipse, se creó un nuevo Modeling Project, éste es de tipo del metamodelo que se creó, en este caso de “arquitecturatesis”. De esa manera se puede generar y comprobar si se generan los objetos del metamodelo correctamente.

La presentación del recurso creado es en forma de estructura de árbol, como muestra la Figura 10, y a los objetos que se crearon en este diagrama se le asignaron sus atributos y propiedades.

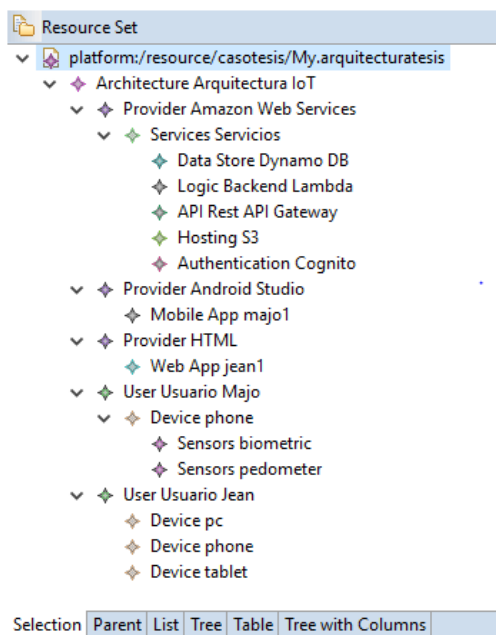


Figura 10 Estructura en forma de árbol

La Figura 10 representa el caso de uso que se pensó para implementar la arquitectura modelada, al encontrarse en forma de árbol, describe todos los elementos escogidos para su desarrollo.

Como se puede observar el sistema consta de tres proveedores, uno para la creación de la página web en HTML, otro para el desarrollo de la aplicación móvil en Android Studio y el último para los servicios en la nube que es AWS. En el lado de usuarios, se definen dos perfiles presentes, el que es parte del nodo IoT, es decir el que

posee el dispositivo con los sensores, y un segundo usuario que es el que puede consultar remotamente los datos almacenados en el servidor con una página web tipo responsive.

Después de haber realizado el DSL en esquema de árbol, para poder realizar pruebas de casos para el metamodelo, se procedió a crear el DSL visual que es mucho más interactivo para el desarrollador de una aplicación basada en la arquitectura propuesta.

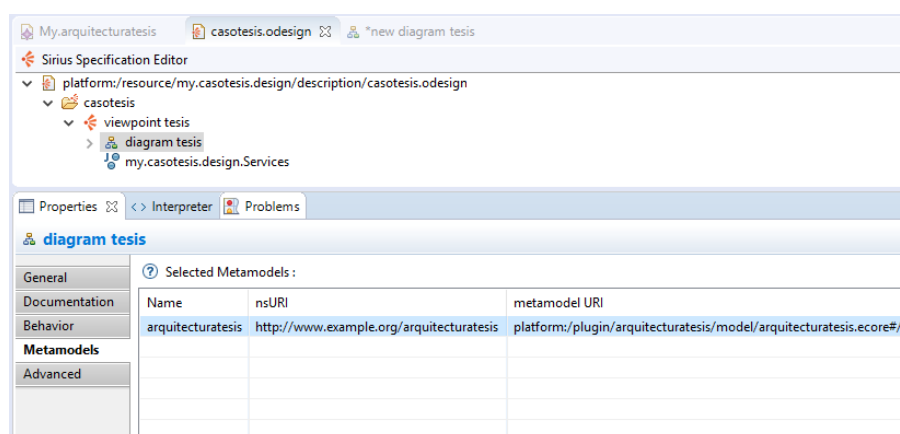


Figura 11 Metamodelo agregado a la representación diagrama

Para el propósito mencionado, se creó un nuevo proyecto de tipo Viewpoint Specification Project, en el cual se diseñaron las herramientas del editor gráfico de modelado. Dentro de este proyecto se creó un viewpoint nombrado “viewpoint tesis” que sirve para crear una representación, que puede ser en forma de diagramas, tablas y árboles. La representación que interesó crear en este modelado fue la de diagrama que se lo denominó “diagrama tesis”. Es importante agregar el metamodelo a la representación,

Figura 11, pues es el que define los objetos que se agregaron a los componentes y herramientas del DSL visual.

En la representación de diagrama de la Figura 12 se agregaron los elementos visuales del DSL, dentro de la capa default se añadieron elementos tipo contenedores, nodos y enlaces.

- Los nodos son los que representen los objetos.
- Los enlaces representan las conexiones entre objetos.
- Los contenedores son útiles para agregar nodos dentro de los mismos.

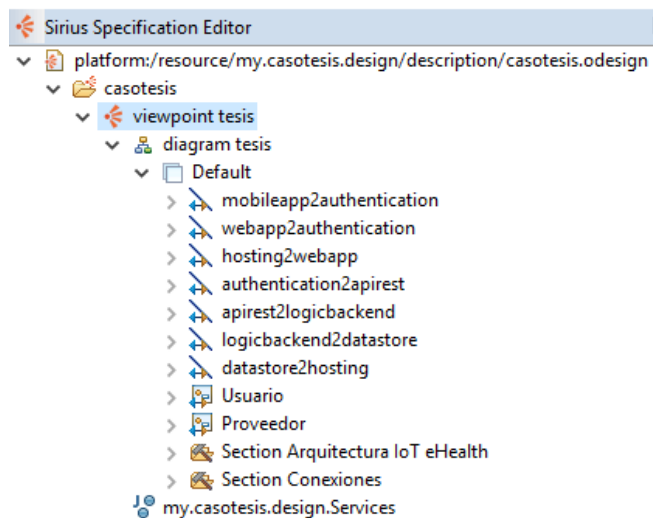


Figura 12 Viewpoint Specification Project

Para la configuración se optó por crear dos contenedores, para objetos de tipo usuario y para objetos de tipo proveedor. En los contenedores, Figura 13, se configuraron las propiedades Id para el nombre del contenedor, domain class para

determinar la clase del metamodelado y semantic candidates expression que indica la ruta para llegar al objeto del metamodelo.

Las propiedades que se configuraron para los nodos son las mismas que para los contenedores.

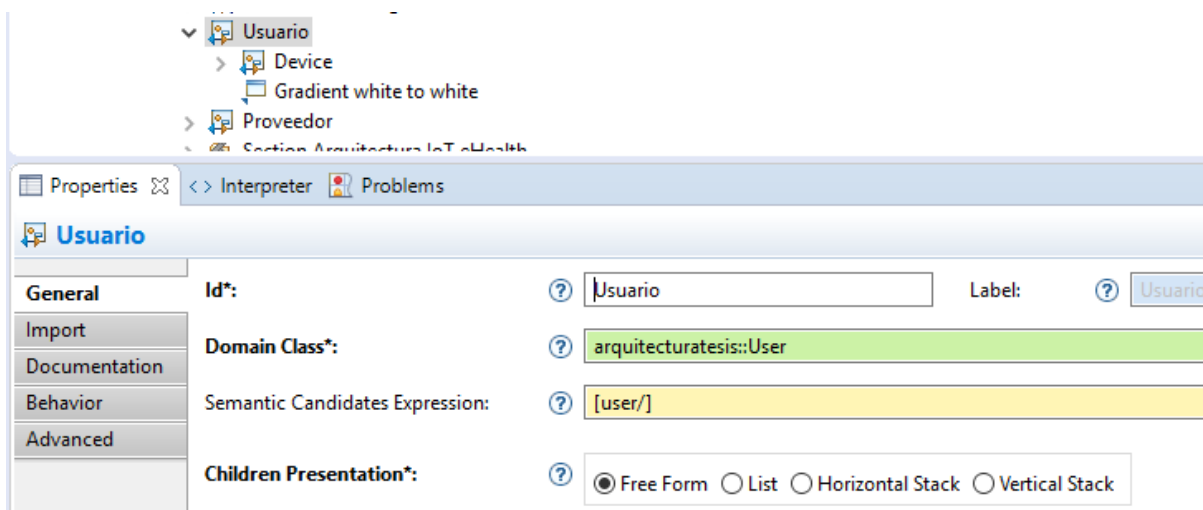


Figura 13 Propiedades de un contenedor

En el contenedor pueden agregarse otros contenedores o nodos, dependiendo la necesidad y el diseño que se requiera. En el caso del contenedor Usuario, Figura 14, se agregó un contenedor tipo Device y siguiendo la jerarquía del metamodelo en el contenedor Device se agregó un nodo tipo Sensor.

Como se puede observar en la misma Figura 14, en el contenedor Device se agregaron estilos condicionales para que al hacer uso del DSL el diseñador de una aplicación basada en el metamodelo planteado, pueda escoger el o los dispositivos. La

misma lógica se aplicó para el nodo Sensor donde se agregaron los estilos condicionales de los sensores definidos.

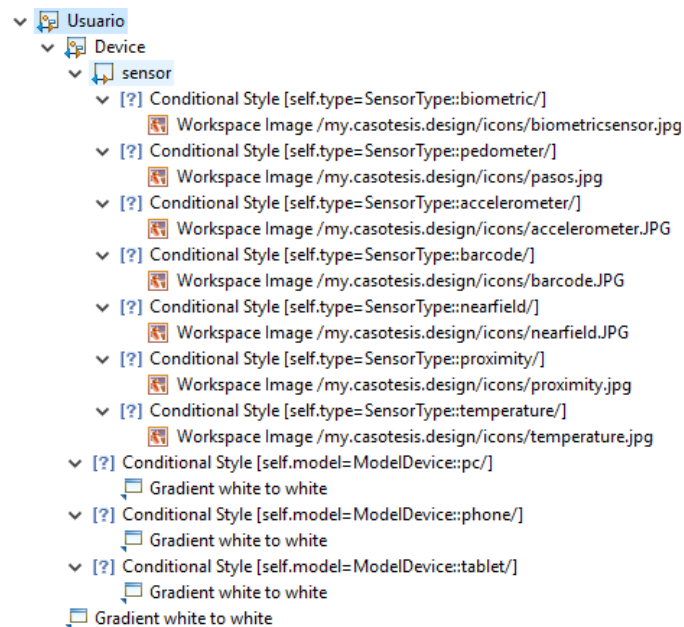


Figura 14 Contenedor Usuario y sus elementos

La única propiedad que se configura en los estilos condicionales es predicate expression, Figura 15, y para su representación visual se creó un elemento workspace image o gradient donde se incluye la imagen respectiva.

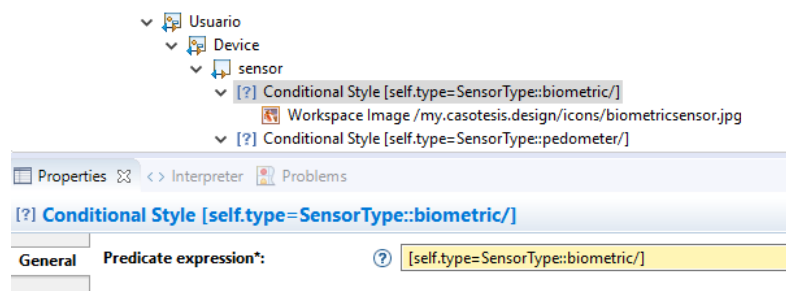


Figura 15 Propiedades de Conditional Style

Con configuraciones similares de las propiedades descritas para los nodos y contenedores, se elaboró la parte que constituyen los objetos de tipo proveedor, Figura 16, en este contenedor se incluyó un nodo Webapp, un nodo Mobileapp, y un contenedor denominado Services. En el contenedor Services se agregaron cinco nodos que representan los objetos correspondientes a authentication, hosting, apirest, logicbackend y datastore.

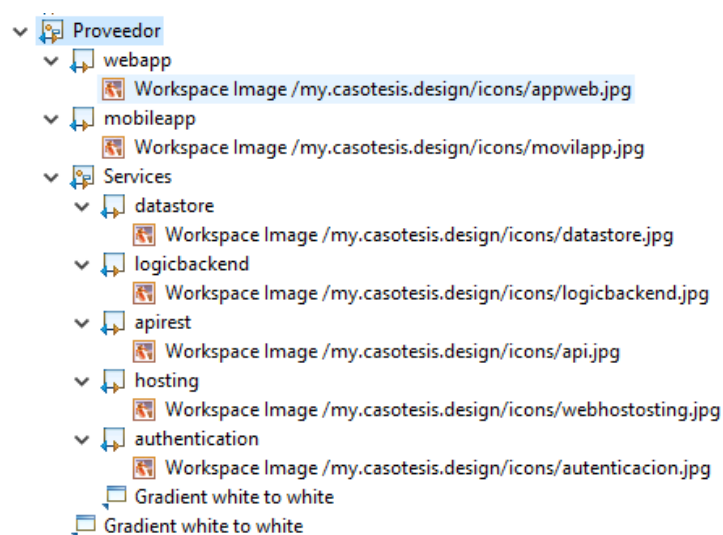


Figura 16 Contenedor Proveedor y sus elementos

Para el caso de los objetos de proveedor no fue necesario la creación de elementos con estilo condicional, simplemente fue suficiente con la representación de una imagen fija para cada objeto.

En cuanto a los enlaces se debió configurar fundamentalmente propiedades como nombre del enlace, nodo origen y nodo destino, Figura 17, se definieron siete enlaces en total.

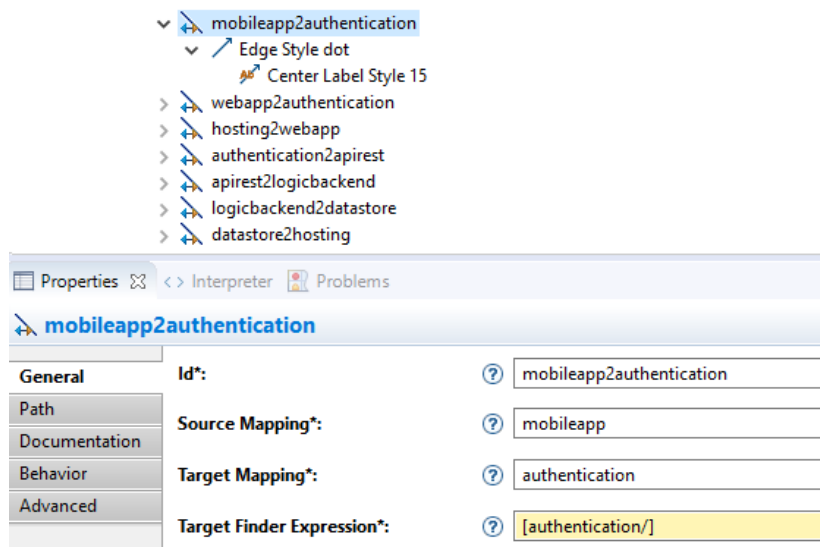


Figura 17 Configuración de propiedades de enlaces

En lo que respecta al diseño de los enlaces, se puede agregar color, estilo y tamaño de letra para una presentación visual más agradable y organizada.

Una vez definidos los elementos visuales, se creó una paleta de herramientas de modelado, con el fin de dar mayor flexibilidad al usuario que desee emplear el DSL. En el mismo viewpoint donde se definieron los objetos del metamodelado, se crearon dos secciones, para los objetos y enlaces que forman parte de la paleta del DSL.

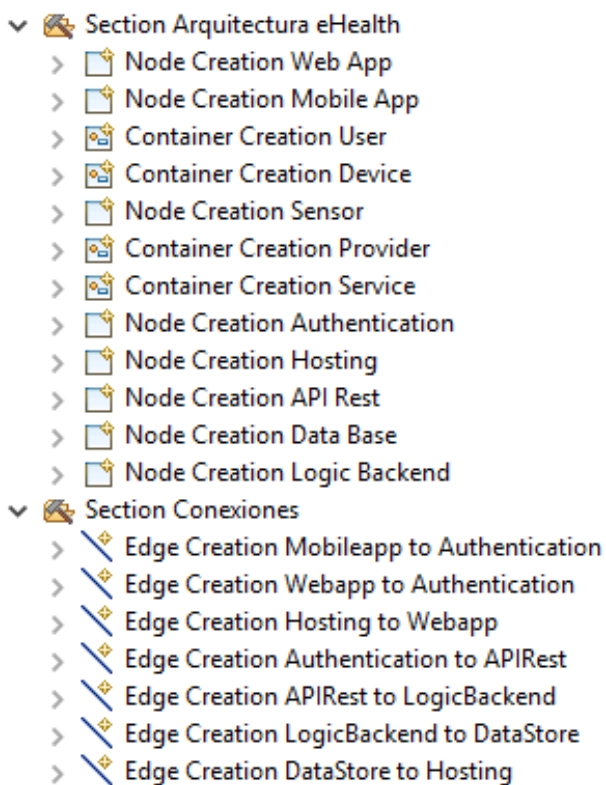


Figura 18 Secciones para la paleta DSL

En la sección de objetos de la arquitectura, se configuraron las propiedades de los nodos y contenedores, Id y el mapeo del nodo correspondiente definido previamente, tal y como se muestra en la Figura 19.

Para definir la acción del nodo o contenedor, se creó una instancia y se relacionó al metamodelado, tal y como se observa en la Figura 20.

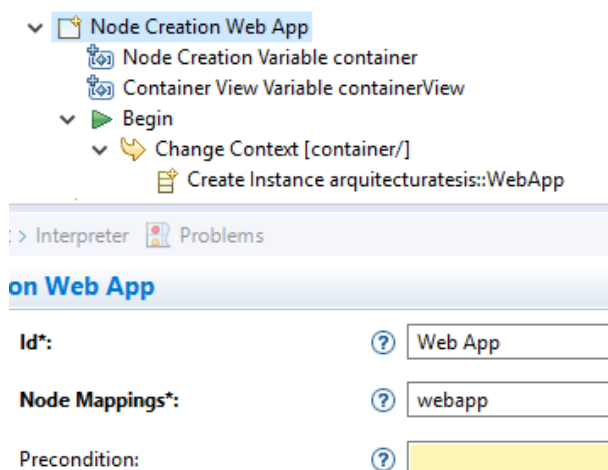


Figura 19 Propiedades de nodos y contenedores

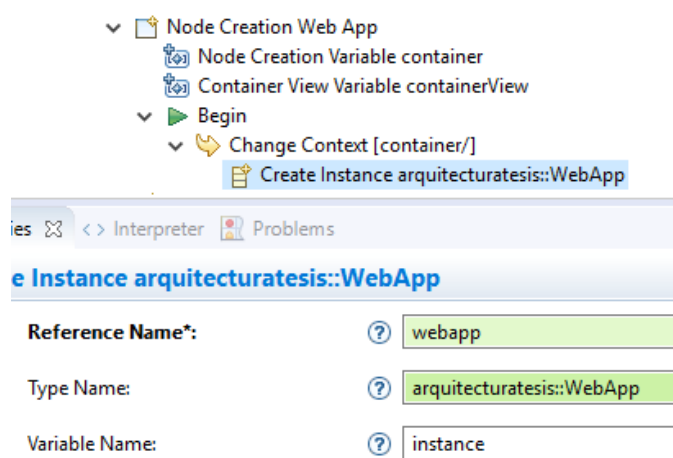


Figura 20 Configuración de acciones de un nodo

En la sección de enlaces de la arquitectura, se configuraron las propiedades de las conexiones entre nodos, Id y el mapeo del enlace correspondiente definido previamente, tal y como se muestra en la Figura 21. Para la definición de la acción del enlace, se creó una operación set y se relacionó con el metamodelado, Figura 22.

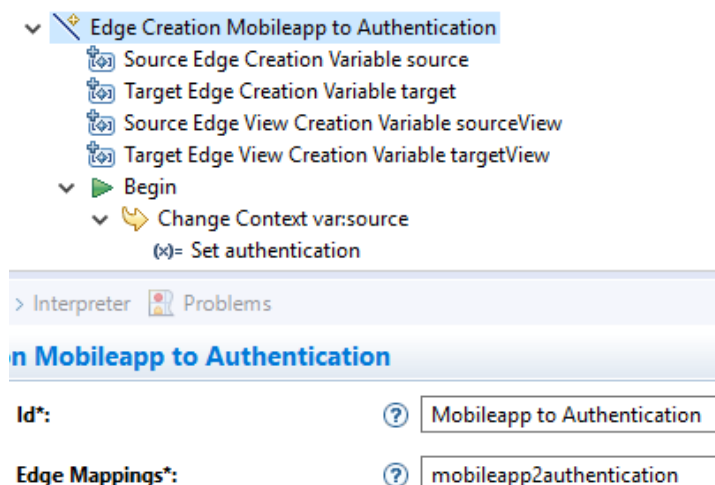


Figura 21 Propiedades de los enlaces

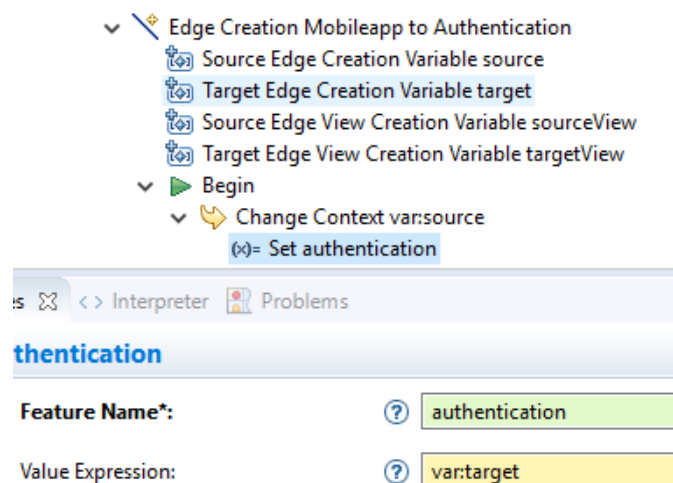


Figura 22 Configuración de acciones de un enlace

La paleta de herramientas del editor visual para el DSL que se creó con las configuraciones mencionadas, se visualiza de la siguiente manera, Figura 23. De igual forma se observa la correspondencia entre los objetos del metamodelo en su representación abstracta y el DSL. La paleta del DSL representa cada objeto del metamodelo abstracto, en la Tabla 7, se detalla cada elemento que define la arquitectura.

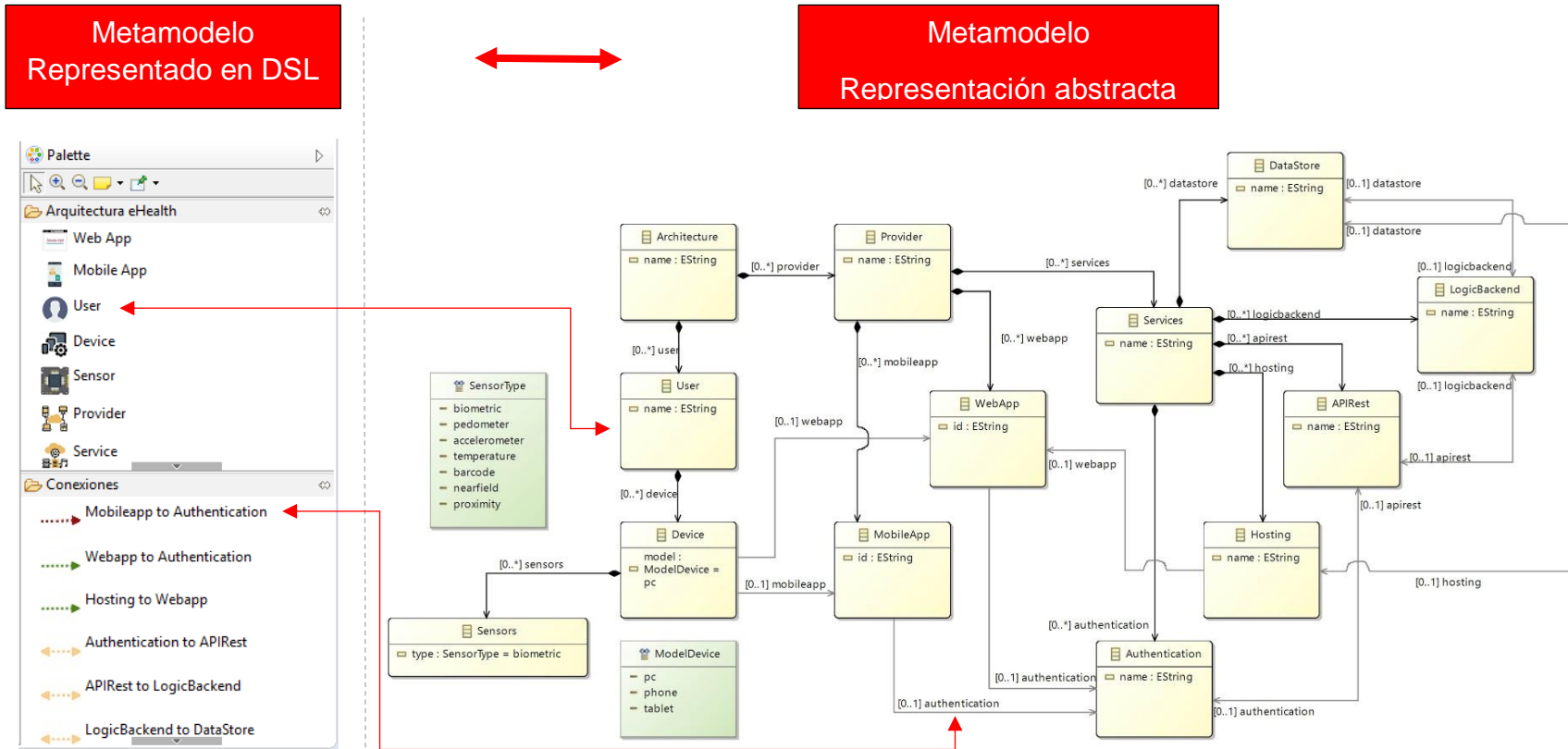





























Figura 23 Metamodelo representado con DSL y metamodelo en sintaxis abstracta

Tabla 7*Relación de correspondencia entre DSL y metamodelo abstracto*

Metamodelo Abstracto	Paleta DSL
Clase User	 User
Clase Provider	 Provider
Clase ModelDevice	 PC
	 Tablet
	 Phone
Clase SensorType	 Biometric
	 Pedometer
	 Temperature
	 Accelerometer
	 NFC
	 Proximity
	 Barcode
Clase MobileApp	 MobileApp
Clase WebApp	 WebApp
Clase Services	 Service
Clase Authentication	 Authentication
Clase Hosting	 Hosting
Clase APIRest	 APIRest
Clase LogicBackend	 LogicBackend
Clase DataStore	 DataStore

Continúa →

Relación WebApp - Authentication		WebApp to Authentication
Relación MobileApp - Authentication		MobileApp to Authentication
Relación Hosting - WebApp		Hosting to WebApp
Relación Authentication - APIRest		Authentication to APIRest
Relación APIRest - LogicBackend		APIRest to LogicBackend
Relación LogicBackend - DataStore		LogicBackend to DataStore
Relación Hosting - DataStore		Hosting to DataStore

Para poder hacer uso del DSL visual, y diseñar aplicaciones basadas en la arquitectura propuesta, se creó una nueva representación del tipo viewpoint que se ha creó en el proyecto de ejemplo “casotesis”, tal y como se observa en la Figura 24.

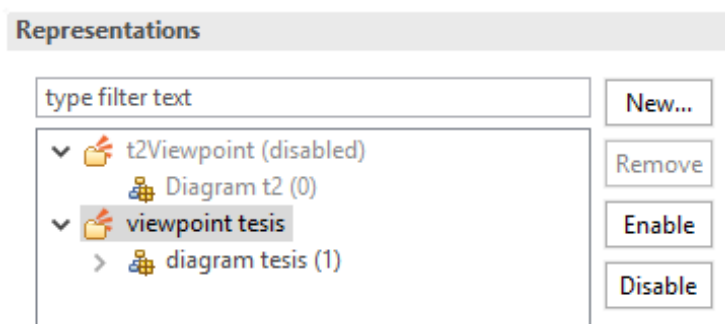


Figura 24 Creación de una representación viewpoint

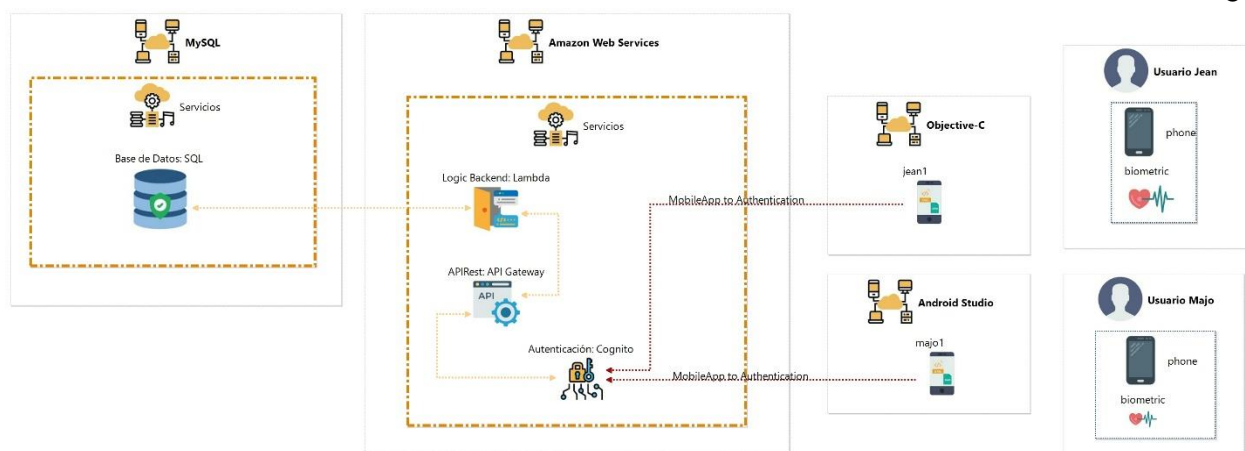


Figura 25 Ejemplo de diseño una aplicación con el DSL

En el DSL creado, ahora se pueden realizar modificaciones, se pueden agregar o quitar elementos y generar un nuevo diseño de aplicación que se encuentre dentro de los parámetros de la arquitectura que se planteó.

Por ejemplo en la Figura 25 se definió un sistema que cuenta con una aplicación móvil para Android e iOS con sensores biométricos, representando al nodo IoT, cuyos datos se almacenan en una base de datos SQL mediante una API RESTful, adicionalmente el sistema posee servicio de autenticación.

CAPÍTULO 4: IMPLEMENTACIÓN DE LA ARQUITECTURA EN EHEALTH

4.1. IMPLEMENTACIÓN

Para implementar la arquitectura desarrollada se realizó el diseño de un caso de uso específico, el DSL de la Figura 26, describe visualmente el sistema desarrollado.

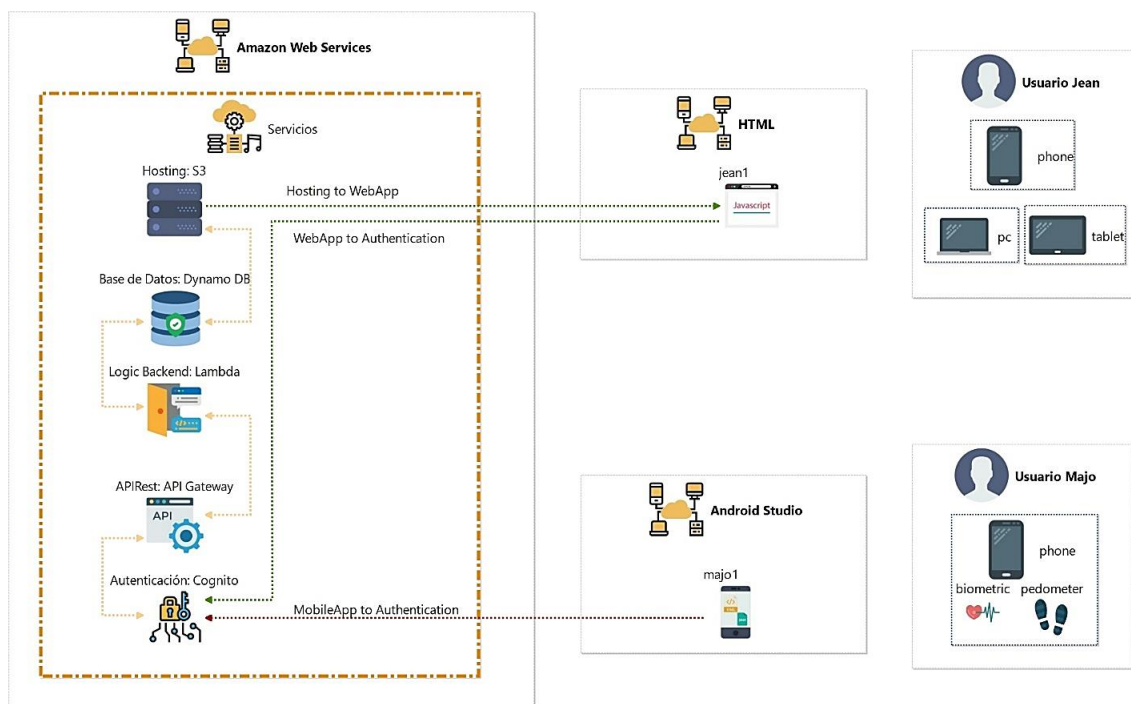


Figura 26 DSL del caso de uso con la aplicación eHealth with AWS

En forma general, el sistema en el lado del cliente es multiusuario, para el caso del nodo IoT se representó mediante un dispositivo móvil y sus sensores de contador de pasos y ritmo cardíaco, para el caso de los usuarios remotos se dispuso de un dispositivo móvil, Tablet o PC que tengan acceso a internet.

El dispositivo móvil que posea los sensores, tiene una aplicación que le permite al usuario poder capturar y almacenar sus datos sensados en una base de datos en la nube de AWS, en cuanto a las consultas remotas se realizan mediante una aplicación web.

Por razones de seguridad y debido a la sensibilidad de datos, el sistema posee el servicio de autenticación tanto en el nodo IoT, como en el de la parte remota.

Con la explicación de las características del sistema y su DSL se definió los recursos de hardware y software para su implementación:

- En cuanto a hardware para el nodo IoT se optó por un teléfono Android, marca Samsung con Sistema Operativo Marshmallow (API 23), y para los sensores se revisó los SDK del contador de pasos y ritmo cardíaco y cámara, que provee Samsung, Tabla 8.
- En cuanto a software, se eligieron programas para el desarrollo de aplicación móvil Android Studio y para la aplicación web el framework de Bootstrap y el editor de texto Brackets. Para el servidor IaaS se seleccionó

Amazon Web Services y los servicios implementados, se presentan en la Tabla 9.

Tabla 8

SDK de sensores en dispositivo Samsung

Sensor	Descripción	Disponibilidad de SDK
Sensor extension	Sensor infrarrojo que mide el	Desarrolladores autorizados
HRM Sensor Signal	ritmo cardíaco.	
Motion	Sensor que mide los pasos	Todos los desarrolladores
Cámara	Procesamiento de imagen	Todos los desarrolladores

Fuente: (Samsung, 2018)

Tabla 9

Servicios AWS utilizados en el sistema

Servicio AWS	Funcionalidad
Amazon Cognito	Autenticación
Amazon DynamoDB	Base de Datos
Amazon API Gateway	API de invocaciones
Amazon Lambda	Funciones de invocación backend
Amazon S3	Hosting

La elaboración del sistema y su implementación se llevó a cabo en dos etapas, la de la aplicación móvil y la de la aplicación web que se detallan a continuación:

4.1.1. Aplicación móvil

Para el desarrollo de la aplicación móvil se eligió un proveedor en el entorno de programación Android Studio 3.1.3, debido a la existencia de mayor documentación de AWS para ésta plataforma dando una mayor flexibilidad de manipulación de código resultando una aplicación más estética y funcional, que la que ofrece MIT ApplInventor.

Para el caso de uso propuesto, el teléfono móvil es el que sensa la actividad cardíaca y los pasos. Mediante el diseño de la app se guardan los datos en la nube de AWS. A continuación se describen las funcionalidades y la configuración que se realizó en el lado del servidor y del cliente.

- **Registro y autenticación**

Para la configuración que se realizó en el lado del servidor se accede a la cuenta AWS Figura 27.

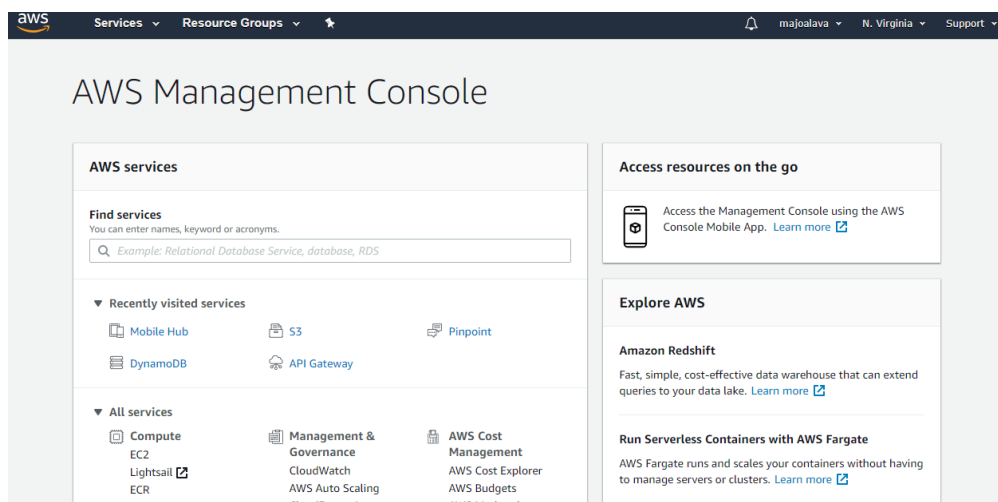


Figura 27 Consola de administrador en AWS

Mediante la consola de AWS se creó un nuevo proyecto tipo Mobile Hub, Figura 28, esto debido a que brinda mayor facilidad para integrar a la configuración de los servicios de autenticación, y base de datos.

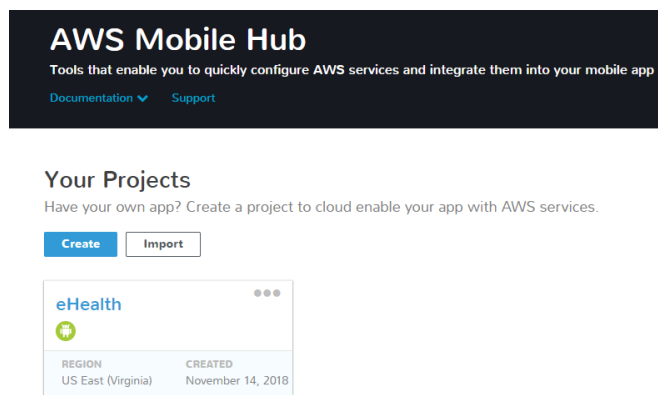


Figura 28 Proyecto eHealth en AWS

Dentro del proyecto, Figura 29, se habilitó por default el servicio de Messaging and Analytics, en este caso no se utiliza ni se realiza ninguna configuración. Los servicios que interesan son Amazon Cognito para la autenticación de usuarios y Amazon DynamoDB para la conexión a la base de datos NoSQL.

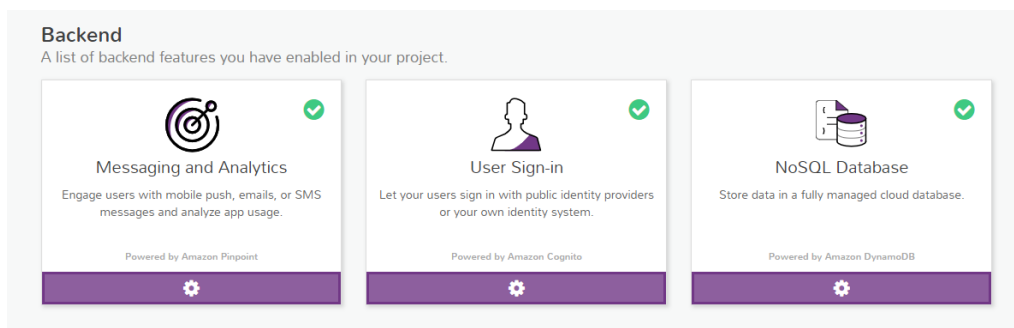


Figura 29 Servicios habilitados en el proyecto de la aplicación móvil

El servicio de Amazon Cognito permite que la aplicación pueda autenticarse mediante correo electrónico, cuenta de Facebook, Google o Mensajes SAML, Figura 30. En esta aplicación se decidió utilizar el correo electrónico el cual debe ser válido, pues el servicio se encarga de enviar un código de confirmación a la cuenta ingresada.

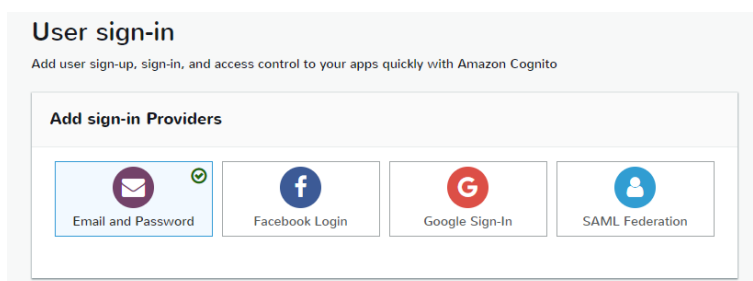


Figura 30 Autenticación por correo electrónico

En la configuración de autenticación mediante correo electrónico, se decidió darle requerimientos a la contraseña, Figura 31, la cual debe tener al menos 6 caracteres y debe contener números y letras. Y el inicio de sesión es mediante un nombre de usuario.

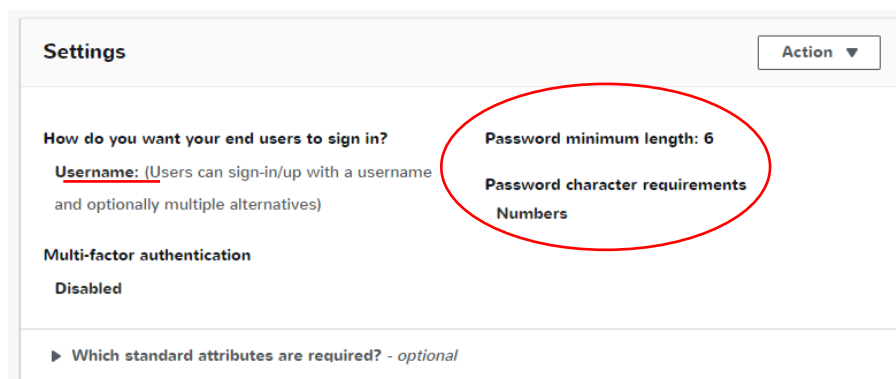


Figura 31 Requisitos para el inicio y cierre de sesión del usuario

En el servicio de Amazon DynamoDB se habilitó una tabla NoSQL, Figura 32, que es la base de datos del sistema, la misma que consta de los atributos que describe la Figura 33.

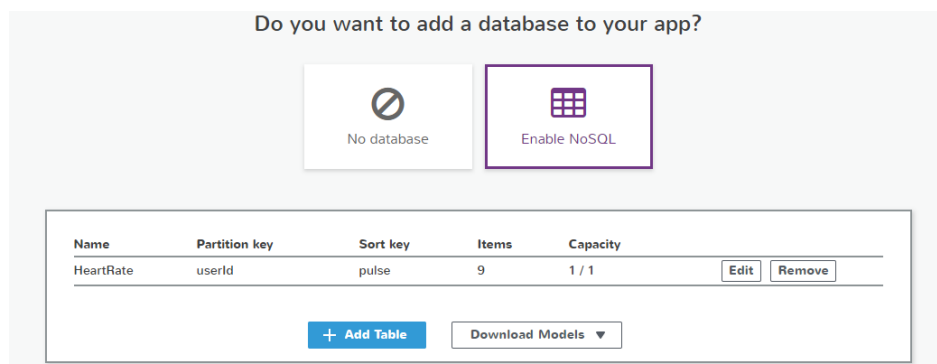


Figura 32 Habilitación de una tabla NoSQL de DynamoDB

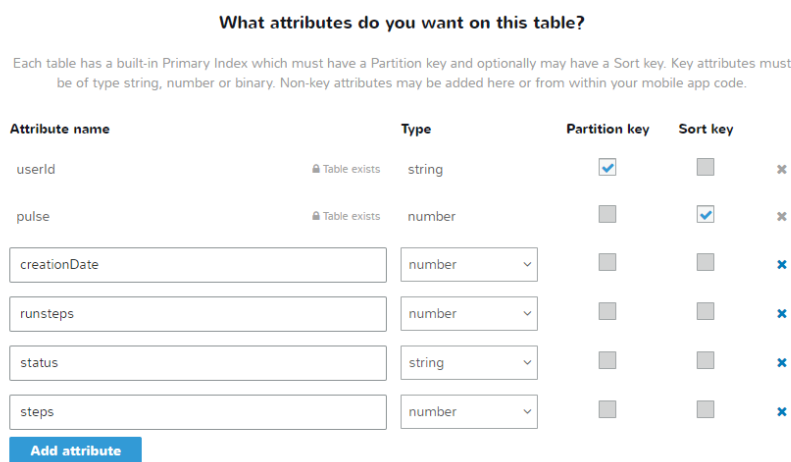


Figura 33 Atributos de la tabla NoSQL del proyecto

Después de la configuración de los servicios en AWS, se la integró a la aplicación que se desarrolló en Android Studio para lo cual Amazon Mobile Hub generó un archivo JSON de configuración de la nube que se debe descargar Figura 34.

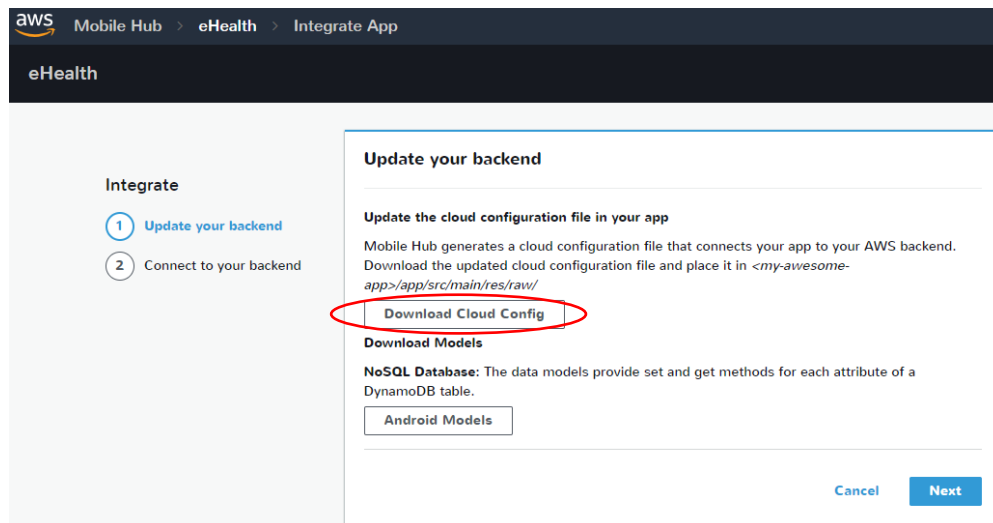


Figura 34 Configuración de la nube para la aplicación en Android

El Proyecto de Android Studio que contiene la aplicación se lo estructuró de la siguiente manera: las clases de Java que poseen la lógica del funcionamiento de las actividades, y los recursos que poseen los archivos de configuración y la interfaz gráfica, Figura 35.

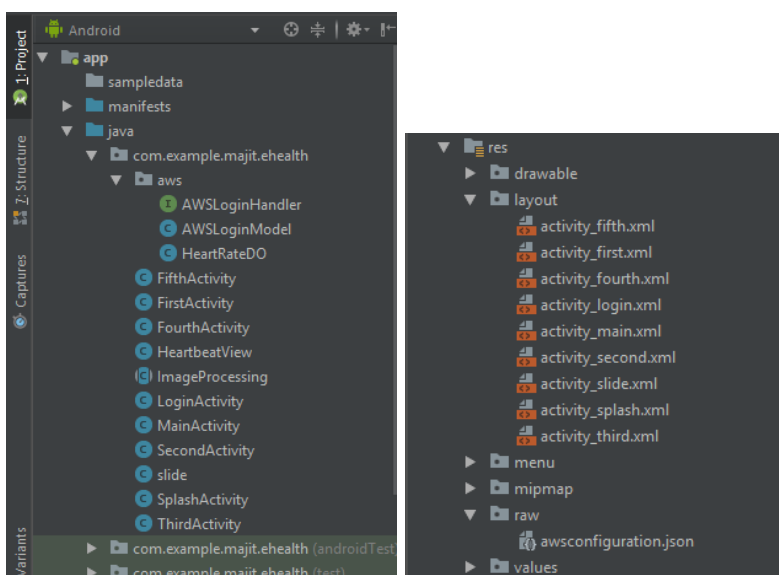


Figura 35 Clases de Java y carpeta de recursos

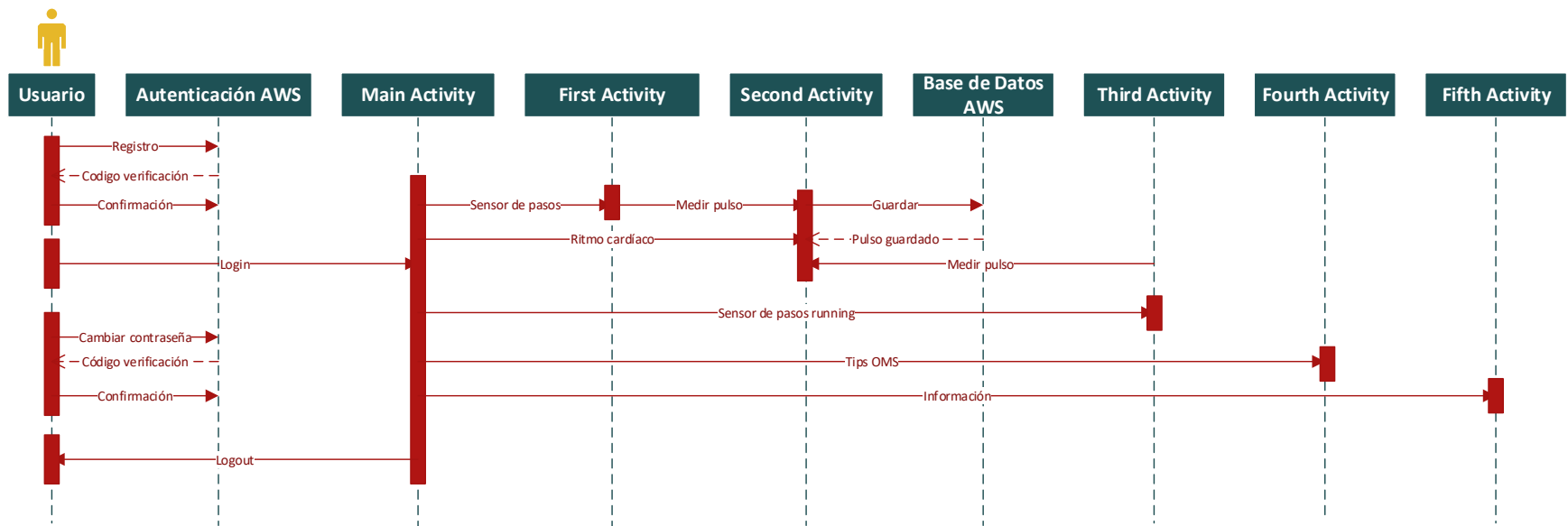


Figura 36 Diagrama de secuencia de actividades del proyecto en Android Studio

La estructura de las clases y los recursos del proyecto, responden a la lógica del diagrama de secuencia entre el usuario y la interacción con la aplicación que se muestra en la Figura 36.

Una vez diseñada y determinada la funcionalidad general de las actividades de la aplicación se procedió a agregar el archivo descargado de AWS al proyecto de Android Studio, Figura 37, su formato es JSON y cada que se realizó un cambio en la configuración de los servicios en Mobile Hub, se volvía a descargar para actualizar a las nuevas configuraciones.

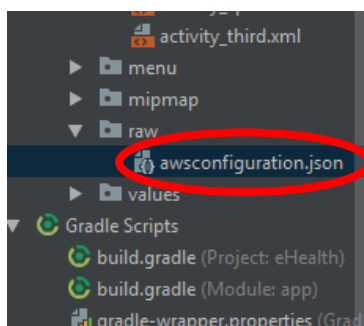


Figura 37 Archivo JSON

En la configuración del lado del cliente con la ayuda de la guía oficial de desarrolladores de AWS se implementaron clases de Java en el proyecto de Android Studio en el que se utilizaron las librerías del SDK proporcionado por el proveedor de AWS. Las librerías se declararon en el archivo build.gradle del proyecto de la siguiente manera:

```
dependencies {  
    implementation fileTree(dir: 'libs', include: ['*.jar'])  
    implementation 'com.android.support:appcompat-v7:28.0.0-alpha3'  
    implementation 'com.android.support.constraint:constraint-layout:1.1.2'
```

```

testImplementation 'junit:junit:4.12'
androidTestImplementation 'com.android.support.test:runner:1.0.2'
androidTestImplementation 'com.android.support.test.espresso:espresso-
core:3.0.2'

//Material Design
implementation 'com.android.support:design:28.0.0-alpha3'
implementation 'com.android.support:cardview-v7:28.0.0-alpha3'
//Mobile Client for initializing the SDK
implementation('com.amazonaws:aws-android-sdk-mobile-client:2.6.+@aar') {
transitive = true; }

//Cognito UserPools for SignIn
implementation 'com.android.support:support-v4:28.+'
implementation('com.amazonaws:aws-android-sdk-auth-userpools:2.6.+@aar') {
transitive = true; }

//Librerías para el podometro de Samsung
implementation files('libs/motion-v2.2.2.jar')
implementation files('libs/sdk-v1.0.0.jar')

//DynamoDB Database NoSQL
implementation 'com.amazonaws:aws-android-sdk-core:2.6.+'
implementation 'com.amazonaws:aws-android-sdk-s3:2.6.+'
implementation 'com.amazonaws:aws-android-sdk-ddb:2.6.+'
implementation 'com.amazonaws:aws-android-sdk-core:2.4.4'
implementation 'com.amazonaws:aws-android-sdk-ddb:2.4.4'
implementation 'com.amazonaws:aws-android-sdk-ddb-document:2.4.4'
implementation 'com.amazonaws:aws-android-sdk-ddb-mapper:2.6.+'
}

```

Con las librerías importadas, para el proceso de autenticación se crearon las instancias de Cognito User Pool, y los métodos necesarios para el registro de un usuario, el inicio y cierre de sesión, y el cambio de contraseña en el proyecto creado.

Finalmente la interfaz gráfica, Figura 38 y Figura 39, que responde a la configuración realizada en AWS y las clases de Java desarrolladas en Android Studio, se visualiza en la aplicación del cliente de la siguiente manera:

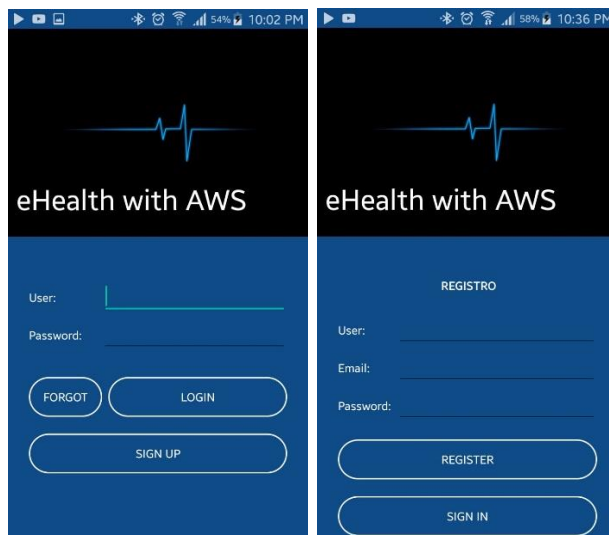


Figura 38 Interfaz gráfica de la autenticación con AWS

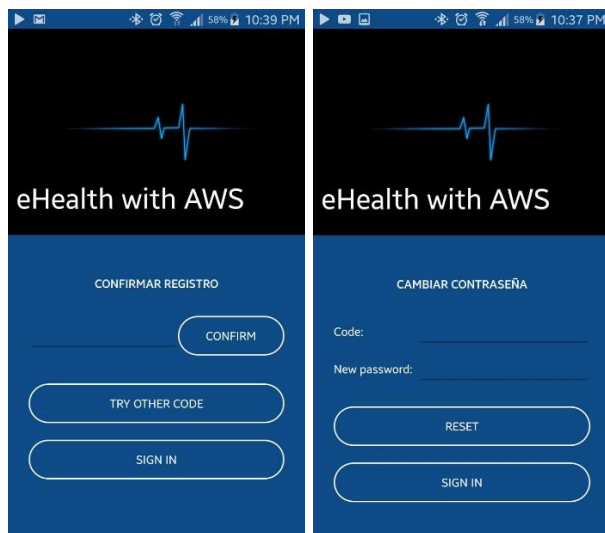


Figura 39 Interfaz gráfica de la autenticación con AWS

- **Menú principal**

En pantalla principal, Figura 40, se muestran algunas funcionalidades de la aplicación, el contador de pasos, el contador de pasos en modo Running, el medidor de

frecuencia cardíaca que es el corazón de la aplicación, una sección de consejos generales para la salud cardiovascular de la OMS (Organización Mundial de la Salud) y la información de la versión de la aplicación.

Esta pantalla principal también cuenta con un menú en la parte superior derecha que permitirá al usuario cerrar sesión si así lo desea.

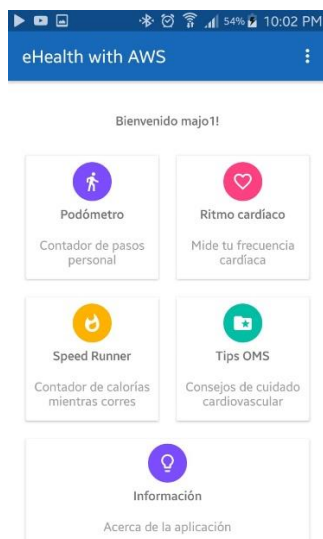


Figura 40 Menú principal

Para la funcionalidad de cada una de las características del menú principal se implementó una clase de Java en Android Studio y su desarrollo se detalla en las siguientes subsecciones.

- **Contadores de pasos**

El contador de pasos se desarrolló haciendo uso del SDK proporcionado por Samsung, funcional a partir de la API 19, o Sistemas Operativos Android 4.4 KitKat en

adelante. Tener el contador de pasos en la misma aplicación permite al usuario realizar su rutina de caminata o running y verificar su pulso cardíaco en ese instante, y guardarlo en la nube.

En Android Studio se instanció el sensor, previamente cargadas las librerías de Smotion y SmotionPedometer de Samsung y se utilizaron los métodos que define el SDK de Samsung en su documentación oficial.

```
protected boolean initializeSensor() {
    motionSensor = new Smotion();

    try {
        motionSensor.initialize(this);
    } catch (SsdkUnsupportedException e) {
        e.printStackTrace();
        return false;
    }

    return motionSensor.isFeatureEnabled(Smotion.TYPE_PEDOMETER) &&
    motionSensor.isFeatureEnabled(Smotion.TYPE_PEDOMETER_WITH_UPDOWN_STEP);
}
```

En las actividades de Podómetro y Speed Runner, Figura 41, se muestra una interfaz gráfica que responde a las necesidades determinadas, muestra en tiempo real los pasos en modo caminata o running, así como las calorías quemadas, distancia y velocidad. En ambas actividades se puede seleccionar el medidor de pulso que llevará al usuario a la siguiente funcionalidad.

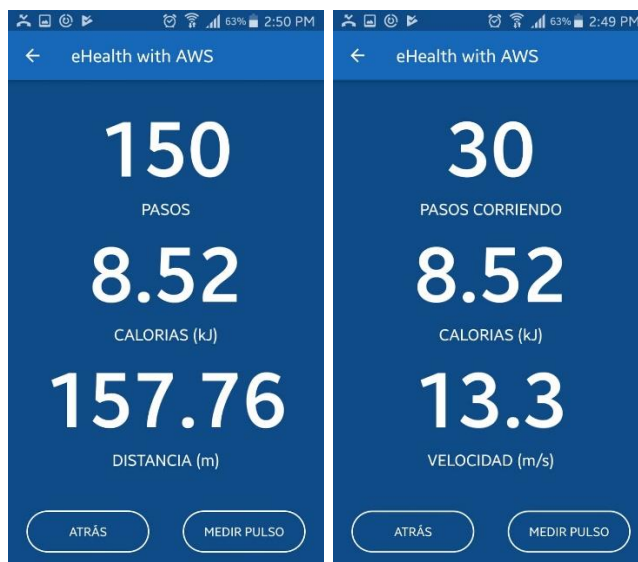


Figura 41 Interfaz gráfica del podómetro

- **Medidor de ritmo cardíaco**

La función principal de la aplicación es el medidor de ritmo cardíaco, pues los datos de ritmo cardíaco, pasos y estado que se guardan en la base de datos DynamoDB se desprenden de esta funcionalidad y son las que se pueden consultar remotamente desde la aplicación web.

En un principio se intentó realizar la medición de los pulsos mediante el SDK de Samsung, disponible desde la API 21, es decir Sistemas Operativos Android 6.0 en adelante y cuyos teléfonos poseen el sensor biométrico HRM para este propósito, sin embargo, el SDK se encuentra disponible solamente para desarrolladores autorizados, y se requieren permisos y aprobaciones de Samsung que no se obtuvo. Por ello la

alternativa fue trabajar con un algoritmo ya desarrollado en una tesis, el cual realiza el procesamiento de los pulsos mediante el flash y la cámara del móvil.

Este algoritmo de procesamiento de imagen propuesto por (Zulhilmi Bin , 2013) utiliza una clase de Java que realiza decodificación de píxeles rojos de la imagen del paso de sangre a través del dedo con la cámara y el flash, dando como resultado el sensor que se requería en este caso de uso. En sus pruebas y resultados verifica la lectura de los pulsos en un rango de 10 a 30 segundos para que sea una medida confiable.

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

<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission
android:name="com.samsung.android.providers.context.permission.WRITE_USE_APP_F
EATURE_SURVEY" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />

<uses-feature android:name="android.hardware.camera" />
<uses-feature android:name="android.hardware.camera.flash" />
```

Debido al uso de la cámara y el flash fue obligatorio declarar en el manifiesto del proyecto los permisos a los recursos del móvil que debe acceder la aplicación.

La interfaz gráfica que se desarrolló en éste proyecto para que responda al propósito descrito se muestra en la Figura 42.

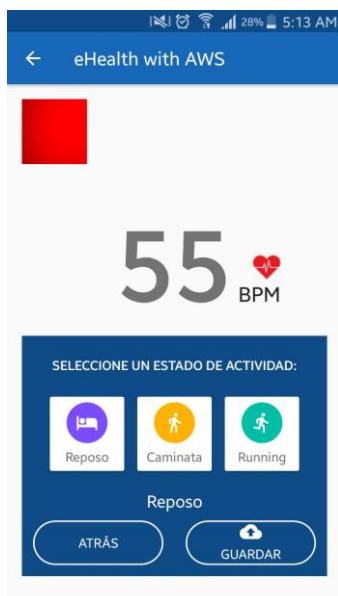


Figura 42 Interfaz gráfica del ritmo cardíaco

En esta actividad se instanció `DynamoDBClient` de la siguiente manera y se obtienen las credenciales de Cognito con el propósito de obtener los permisos de acceso.

```
AmazonDynamoDBClient dynamoDBClient = new
AmazonDynamoDBClient(AWSMobileClient.getInstance().getCredentialsProvider());
this.dynamoDBMapper = DynamoDBMapper.builder()
    .dynamoDBClient(dynamoDBClient)
    .awsConfiguration(AWSMobileClient.getInstance().getConfiguration())
    .build();
```

Luego en el método que proporciona AWS que realiza el almacenamiento de datos se creó un objeto de la tabla DynamoDB y se ejecuta la operación en un hilo como recomienda la documentación de AWS para Android.

```
public void createPulso() {

    final HeartRateDO newsItem = new HeartRateDO();

    String userName = AWSLoginModel.getSavedUserName(SecondActivity.this);
    String status = (String) text_status.getText();
    String pulso = (String) text.getText();
    long mydate = System.currentTimeMillis();
```



```
newsItem.setUserId(userName);
newsItem.setStatus(status);
newsItem.setPulse(Double.parseDouble(pulso));
newsItem.setCreationDate((double) mydate);
newsItem.setSteps(stepswalk);
newsItem.setRunsteps(stepsrun);

new Thread(new Runnable() {

    @Override
    public void run() {
        dynamoDBMapper.save(newsItem);
        // Item saved
    }
}).start();
}
```

- **Tips OMS e información**

En ésta actividad, se realizó un resumen interactivo con slide de las principales recomendaciones de la OMS para el cuidado de la salud cardiovascular. Cabe destacar que son recomendaciones generales para la prevención de enfermedades cardíacas, consideradas de Categoría primaria para la OMS descrita en (OMS, 2008).

La interfaz gráfica que responde a esta funcionalidad se presenta en la Figura 43. Con fines de brindar confiabilidad al usuario en la actividad de Información se muestran detalles de la aplicación y número de versión.

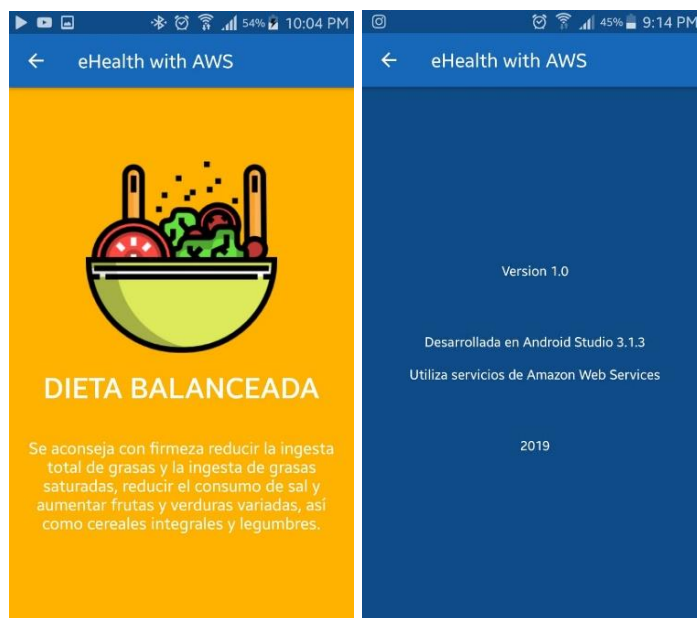


Figura 43 Actividad Tips OMS e Información

4.1.2. Aplicación web

Para el desarrollo de la aplicación web, se determinó que debía ser multidispositivo, es decir debe ser una página web con un diseño totalmente responsive para que se adapte a todos los tamaños de pantallas de los dispositivos, desde móviles hasta laptops.

Para cumplir el requisito de diseño responsive se acudió a los diseños de Bootstrap 4.1.3 que es un kit de desarrollo de código abierto para trabajar con HTML, CSS y JS con este propósito.

- **Página web**

Para realizar la aplicación web, primeramente se alojó mediante la consola de AWS todo el contenido estático de la página web desarrollada en el servicio Amazon S3, pues de esta manera Amazon proporciona la URL, Figura 44, donde se muestra la página web desde cualquier navegador, la URL puede ser personalizada sin embargo, por cuestión de costos se dejó la predeterminada.

En posteriores subsecciones la página se muestra de forma dinámica, esto será mediante la adición de los archivos JavaScript con JQuery y Ajax que se le agregarán para que exista la debida interacción con el usuario.

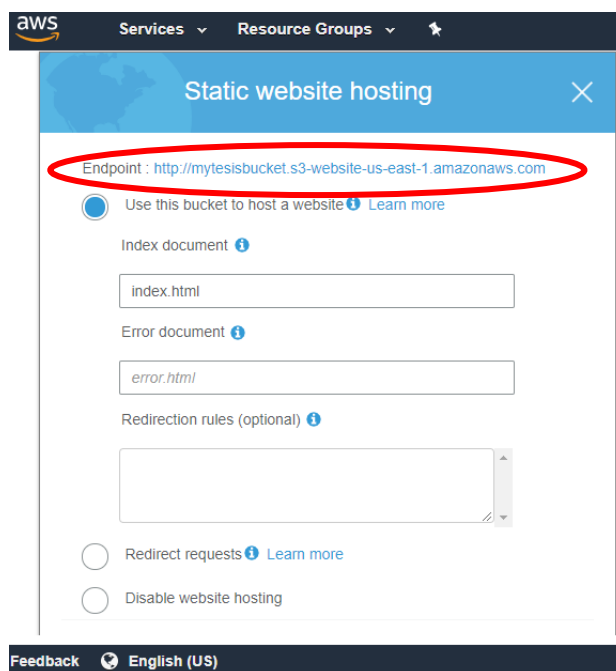


Figura 44 Hosting web y la URL pública

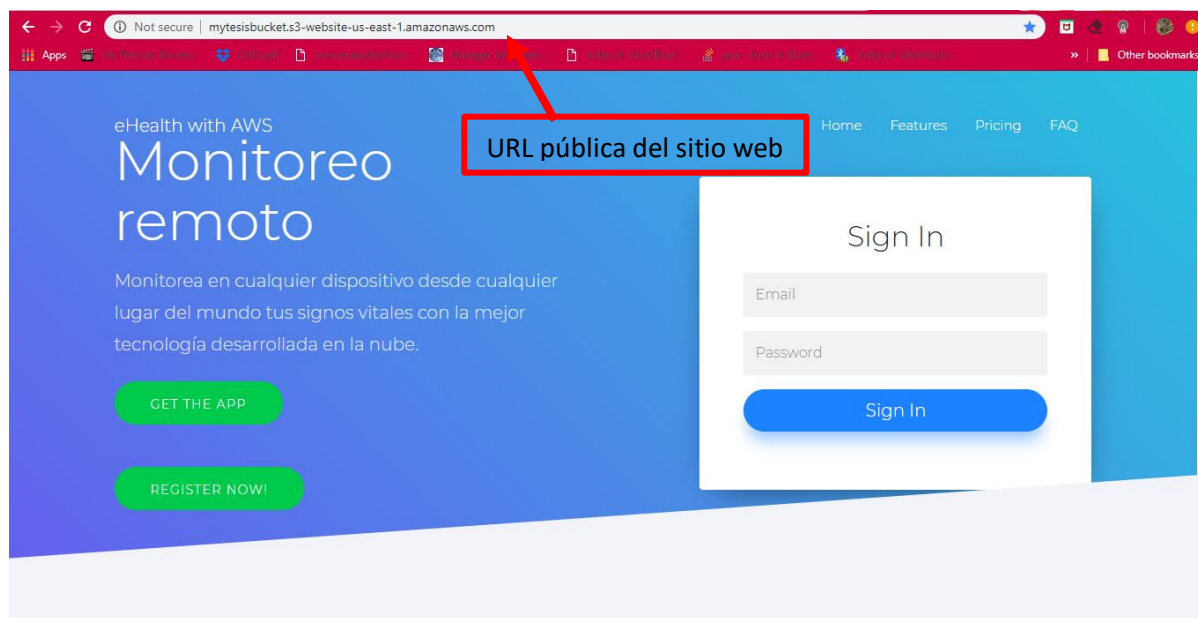


Figura 45 Página web alojada mediante Amazon S3 en una URL pública

- **Registro y autenticación**

Para la aplicación web es indispensable la autenticación de usuarios, por seguridad, y nuevamente haciendo uso del servicio de Amazon Cognito, esta vez aplicado mediante JavaScript que realiza el proceso de registro mediante correo electrónico válido y un código de confirmación que proporciona Amazon.

En el lado del servidor Amazon Cognito, se creó un grupo de usuarios, llamado ehealthclients, se le dió las características de los atributos requeridos como email para este caso y los requisitos de contraseña que sea de mínimo 6 caracteres con números y letras. El resumen del grupo de usuarios creados se muestra en la Figura 46.

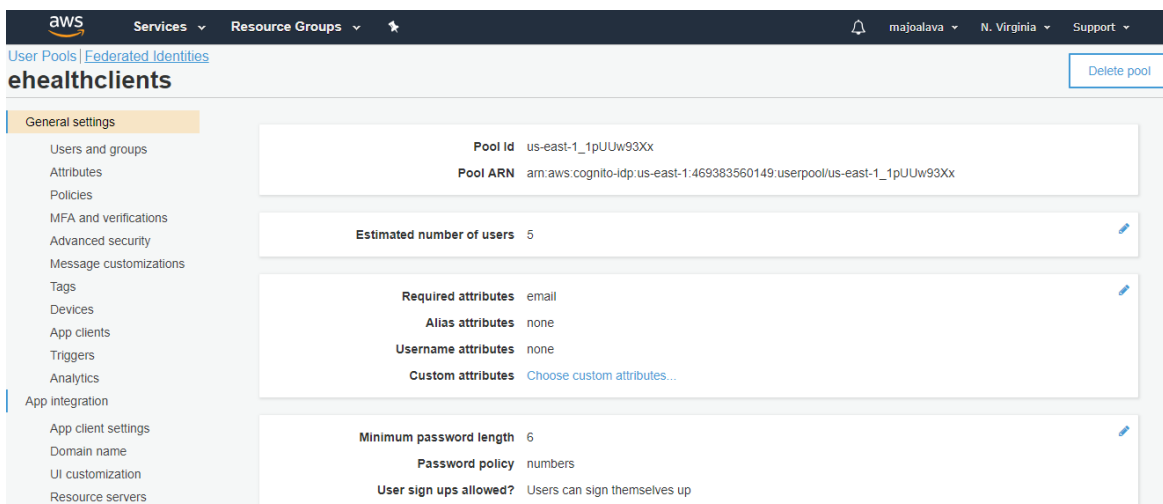


Figura 46 Resumen de características del grupo de usuarios para la página web

Con el recurso configurado, se creó un archivo de JavaScript donde se declara un objeto `CognitoUserPool` con su `UserPoolId` y el `ClientId`, que son parámetros que entrega AWS en la configuración del servicio.

```

window._config = {
  cognito: {
    userPoolId: 'us-east-1_1pUUw93Xx',
    userPoolClientId: '7u8ars1blv7u2ecsm2gbeo9eud',
    region: 'us-east-1'
  },
  api: {
    invokeUrl: 'https://i5wa4r3ixj.execute-api.us-east-1.amazonaws.com/prod1'
  }
};

```

Luego se determinaron las funciones para el registro de usuarios, la verificación mediante correo electrónico y el inicio de sesión en base a la documentación del SDK que proporciona AWS para realizar con éxito éstas acciones fundamentales en el proyecto.

- **Creación de la API RESTful**

La API RESTful que se elaboró tiene el objetivo de realizar solicitudes HTTP GET a la base de datos de DynamoDB, debido a las políticas de seguridad para acceder a datos sensibles y personales en aplicaciones de salud, fue necesario que su gestión posea autorización caso contrario deniega el acceso. Con éste propósito se procedió a configurar los servicios de AWS pertinentes para lograr dicha funcionalidad de la aplicación web.

Se utilizó el servicio de AWS Lambda debido a la característica fundamental de ejecutar una aplicación en el servidor pero sin tener que preocuparse por tareas de administración del mismo.

The screenshot displays the AWS IAM console interface for the 'ehealthLambda' role. The 'Summary' tab is active, showing the following details:

- Role ARN:** `arn:aws:iam::469383560149:role/ehealthLambda`
- Role description:** Allows Lambda functions to call AWS services on your behalf.
- Instance Profile ARNs:** (None listed)
- Path:** /
- Creation time:** 2018-11-15 22:51 EST
- Maximum CLI/API session duration:** 1 hour

The 'Permissions' tab is also visible, showing that one policy is applied: 'AWSLambdaBasicExecutionRole', which is an AWS managed policy.

Figura 47 Rol de IAM para la función de invocación de la API

Para la configuración de éste servicio de funciones de invocación se determinó un rol de IAM, éste define los servicios de AWS con los cuales la función puede interactuar.

Como se requería una conexión con la base de datos NoSQL de DynamoDB se adjuntó políticas de permiso de lectura y escritura a la función para acceder a la información de la tabla, como indica la Figura 48.

Review policy

Name* DynamoDBAccess
Use alphanumeric and '+', '@', '_' characters. Maximum 128 characters.

Description Acceso a la base de datos Dynamo DB
Maximum 1000 characters. Use alphanumeric and '+', '@', '_' characters.

Summary

Filter

Service	Access level	Resource	Request condition
Allow (1 of 148 services) Show remaining 147			
DynamoDB	Limited: Read, Write	TableName string like ehealth-mobilehub-409083068-HeartRate	None

Figura 48 Política de acceso de la función a DynamoDB

Summary

Role ARN: [arn:aws:iam::469383560149:role/ehealthLambda](#)

Role description: Allows Lambda functions to call AWS services on your behalf. [| Edit](#)

Instance Profile ARNs: [| Edit](#)

Path: /

Creation time: 2018-11-15 22:51 EST

Maximum CLI/API session duration: 1 hour [| Edit](#)

Permissions | Trust relationships | Tags | Access Advisor | Revoke sessions

Permissions policies (2 policies applied)

[Attach policies](#)

Policy name	Policy type
AWSLambdaBasicExecutionRole	AWS managed policy
DynamoDBAccess	Inline policy

Figura 49 Rol de IAM con la política de acceso a DynamoDB

Como se puede observar en Figura 49, el Rol de IAM para la función ya posee los permisos configurados de acceso a la tabla de DynamoDB del proyecto.

En el detalle de la política, Figura 50, se puede observar que es un archivo JSON y define las acciones que se pueden utilizar en la función de invocación, GetItem, PutItem, Scan y Query.

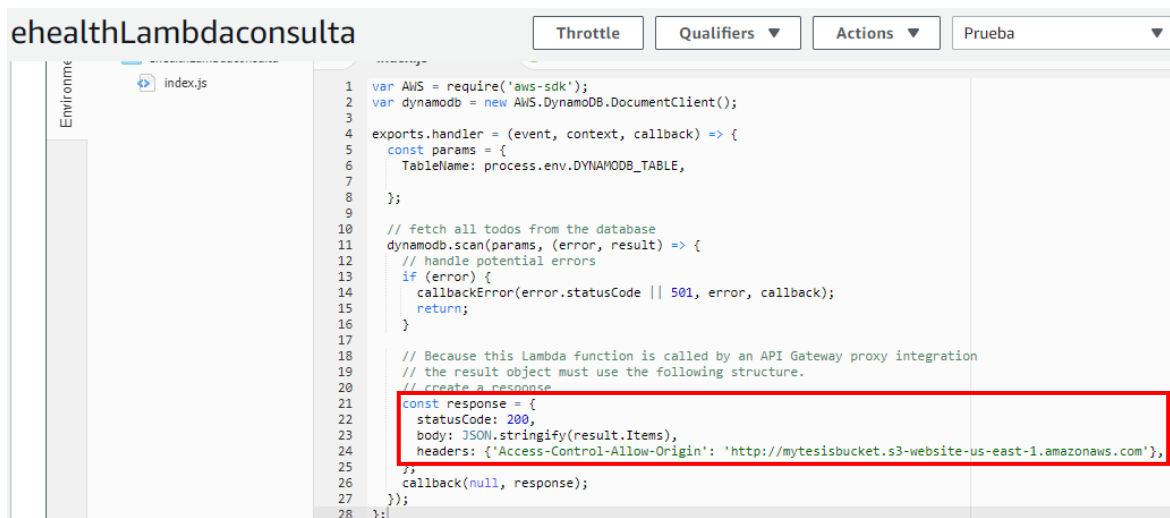


Figura 50 Política de las acciones permitidas para DynamoDB

Con los permisos configurados, se creó una función de invocación en AWS Lambda, en este servicio se tiene una consola que permite la creación del archivo en lenguaje de JavaScript. En este archivo se programó una acción “scan” de la tabla del proyecto en Amazon DynamoDB y al invocarla entrega a la API un archivo JSON con todos los objetos que se encuentren almacenados en la misma.

Adicionalmente a los objetos que retorna la función, se agregó un statusCode 200 que indica que la solicitud se realizó con éxito y un header de respuesta CORS (Cross-Origin Resource Sharing), que es una solicitud de verificación previa para la

comunicación entre dominios con el recurso de DynamoDB, con este paso se aseguró dar un control de acceso restringido y con mayor seguridad de los datos.



```

1 var AWS = require('aws-sdk');
2 var dynamodb = new AWS.DynamoDB.DocumentClient();
3
4 exports.handler = (event, context, callback) => {
5   const params = {
6     TableName: process.env.DYNAMODB_TABLE,
7   };
8 };
9
10 // fetch all todos from the database
11 dynamodb.scan(params, (error, result) => {
12   // handle potential errors
13   if (error) {
14     callbackError(error.statusCode || 501, error, callback);
15     return;
16   }
17
18   // Because this Lambda function is called by an API Gateway proxy integration
19   // the result object must use the following structure.
20   // create a response
21   const response = {
22     statusCode: 200,
23     body: JSON.stringify(result.Items),
24     headers: {'Access-Control-Allow-Origin': 'http://mytesisbucket.s3-website-us-east-1.amazonaws.com'},
25   };
26   callback(null, response);
27 });
28 };

```

Figura 51 Función de invocación con seguridad CORS

Para verificar la función, la consola de AWS Lambda, permite realizar el testeo de la misma, y su resultado se visualizó de la siguiente forma:



```

Execution Result: x
Status: Succeeded Max Memory Used: 37 MB Time: 1052.07 ms
Response:
{
  "statusCode": 200,
  "body": "[{\\"steps\\":451,\\\"creationDate\\":1547266708694,\\\"pulse\\":53,\\\"userId\\":\\\"majo1\\\",\\\"runsteps\\":149,\\\"status\\":\\\"Reposo\\\"},",
  "headers": {
    "Access-Control-Allow-Origin": "http://mytesisbucket.s3-website-us-east-1.amazonaws.com"
  }
}

```

Figura 52 Testeo de la función de invocación exitosa

Con la función de invocación configurada correctamente, se procedió a crear la API que invocará dicha función en el servicio de Amazon API Gateway, Figura 53.

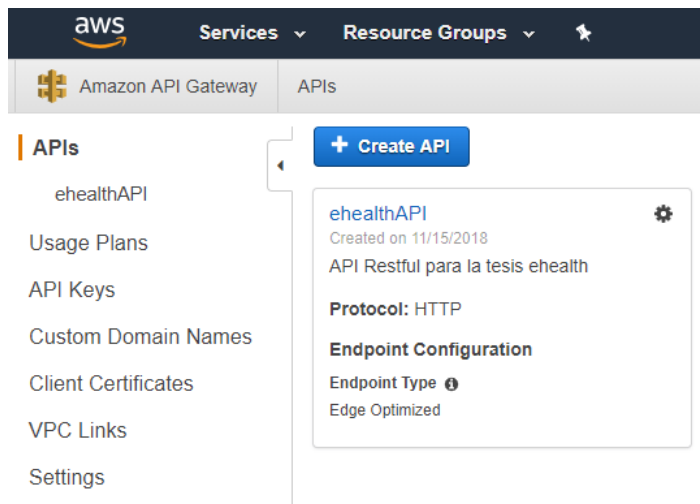


Figura 53 API de invocación de la función lambda

El método GET que se implementa en la API emplea otro proceso de seguridad de datos mediante la obtención del Token de acceso del grupo de usuarios de Amazon Cognito, éste es el autorizador.

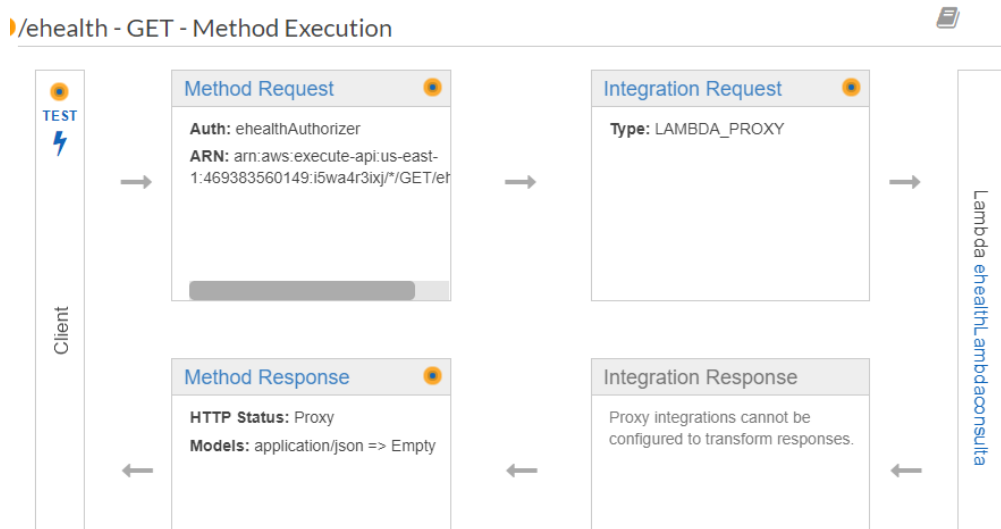
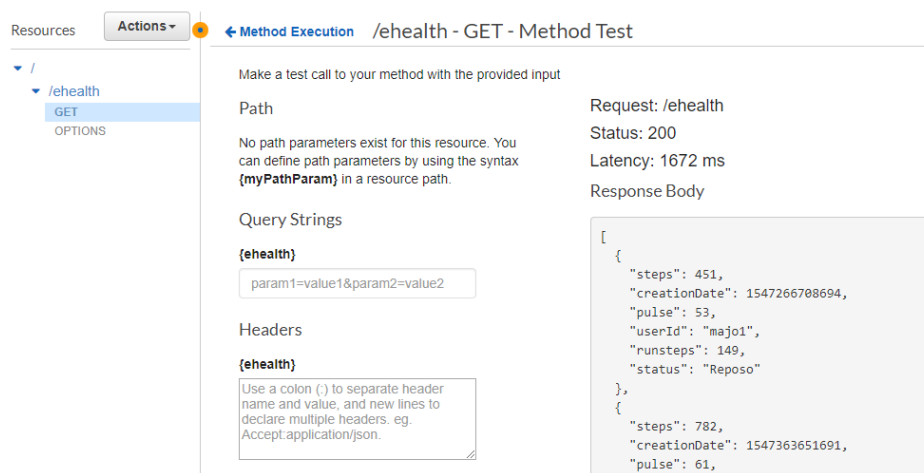


Figura 54 Método GET y el resumen de configuración y flujo de ejecución

Se comprueba el funcionamiento de la API, y se observa que la invocación se realiza correctamente devolviendo el JSON con los datos y encabezado CORS determinados en la función, Figura 55.

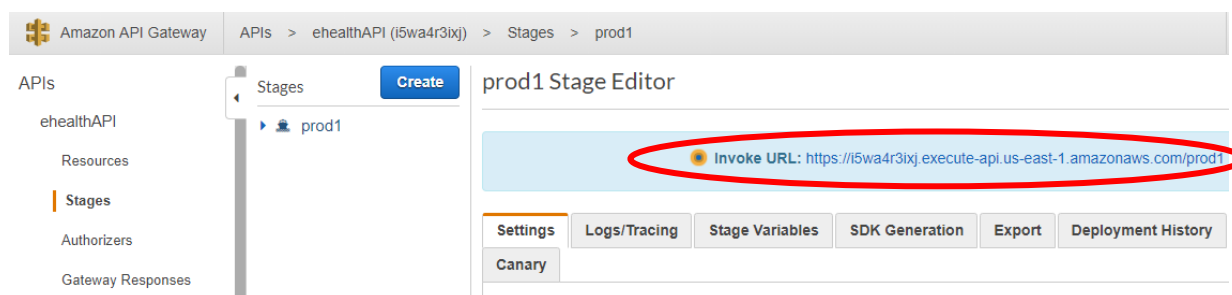


The screenshot shows the 'Method Execution' page for the GET method on the /ehealth resource. The 'Request' section shows the path /ehealth, status 200, and latency of 1672 ms. The 'Response Body' section displays a JSON array of two objects:

```
[
  {
    "steps": 451,
    "creationDate": 1547266708694,
    "pulse": 53,
    "userId": "majoi",
    "runsteps": 149,
    "status": "Repo"
  },
  {
    "steps": 782,
    "creationDate": 1547363651691,
    "pulse": 61,
  }
]
```

Figura 55 Test de la API que invoca la función lambda

Finalmente, se desplegó la API y se obtuvo la URL de invocación, Figura 56, que se utiliza en los archivos de JavaScript de la página web y mediante peticiones AJAX se crea y actualiza el contenido que se consulta dinámicamente.



The screenshot shows the 'prod1 Stage Editor' page in the Amazon API Gateway console. The 'Invoke URL' is highlighted with a red oval, showing the URL: <https://i5wa4r3ixj.execute-api.us-east-1.amazonaws.com/prod1>. The page also shows various settings and options for the stage.

Figura 56 Obtención de la URL de invocación de la API

```

$(function onDocReady() {
    var ultimox;
    var ultimoy;
    setInterval(function () {
        $.ajax({
            url: 'https://i5wa4r3ixj.execute-api.us-east-
1.amazonaws.com/prod1/ehealth',
            method: "GET",
            headers: {
                Authorization: authToken
            },
            crossDomain: true,
            contentType: 'application/json',
            dataType: "json",
            success: function(result){
                //ordeno los datos por fecha
                result.sort(function (a, b) {
                    return (a.creationDate - b.creationDate)
                })
                console.log('Datos ordenados: ', result);
                var datareal = [];
                for (var i in result){
                    var auxserie = new Array(result[i].userId, result[i].pulse);
                    datareal.push(auxserie);
                }
                console.log('Datareal: ', datareal);

                $.each(datareal, function(i,o){
                    if (o.x) {datareal[i].x = parseFloat(o.x);}
                    if (o.y) {datareal[i].y = parseInt(o.y);}
                });
                DibujaGraficoReal(datareal);
                //console.log('Datareal: ', datareal);
                var varlocalx=parseFloat(datareal[0].x);
                var varlocaly=parseFloat(datareal[0].y);
                if((getx() !=varlocalx) && (gety() !=varlocaly)){
                    series.addPoint([varlocalx, varlocaly], true, true);
                    setx(varlocalx);
                    sety(varlocaly);
                }
                setx(datareal[(datareal.length)-1].x);
                sety(datareal[(datareal.length)-1].y);
            }
        });
    }, 60000);

    function getx(){return ultimox;}
    function gety(){return ultimoy;}
    function setx(x){ultimox=x;}
    function sety(y){ultimoy=y;}

```

La página web finalmente tiene los permisos configurados correctamente para el despliegue de la consulta de datos remotamente. Para demostrar que se obtiene acceso

correcto en las herramientas de desarrollador del navegador se visualiza en consola los logs de comprobación, Figura 57.

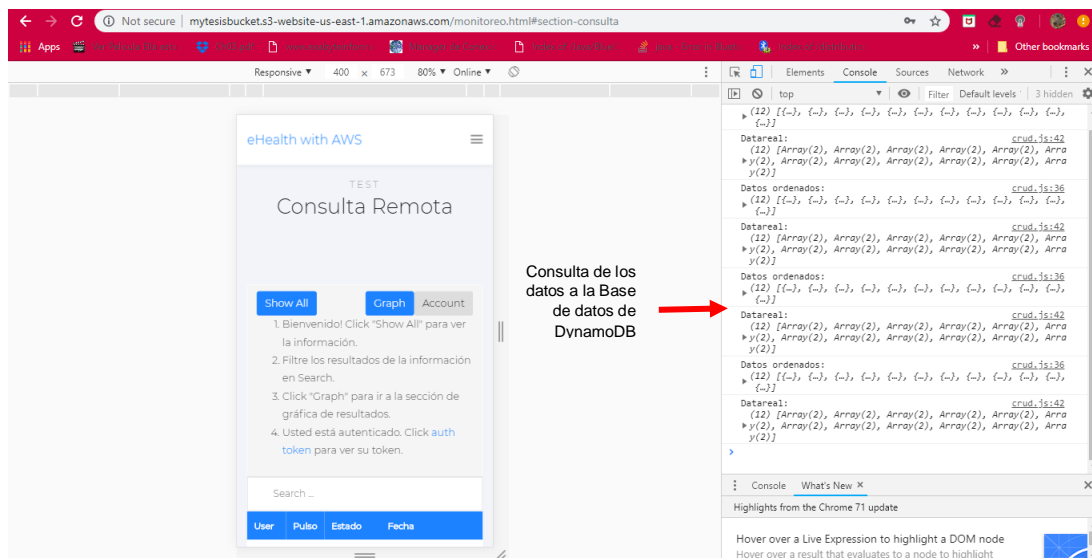


Figura 57 Consulta exitosa, a través del dominio correcto para CORS

La seguridad implementada con CORS, se comprueba al intentar acceder mediante otro dominio de origen que no se especifica en la función de invocación, Figura 58.

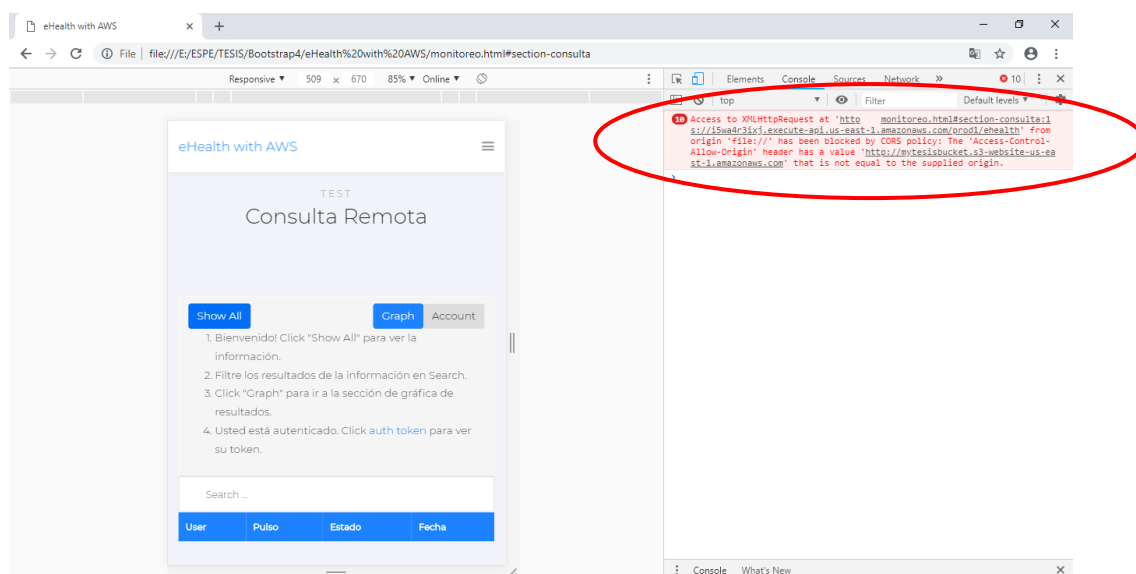


Figura 58 Acceso denegado, a través del dominio incorrecto para CORS

CAPÍTULO 5: PRUEBAS Y RESULTADOS

5.1. PRUEBAS DE ARQUITECTURA

Utilizando la herramienta del DSL que se creó para diseñar aplicaciones de eHealth con la arquitectura propuesta, se propone un escenario de prueba en el que se agrega un proveedor de servicio de hosting y se integra con el proveedor de los demás servicios en AWS.

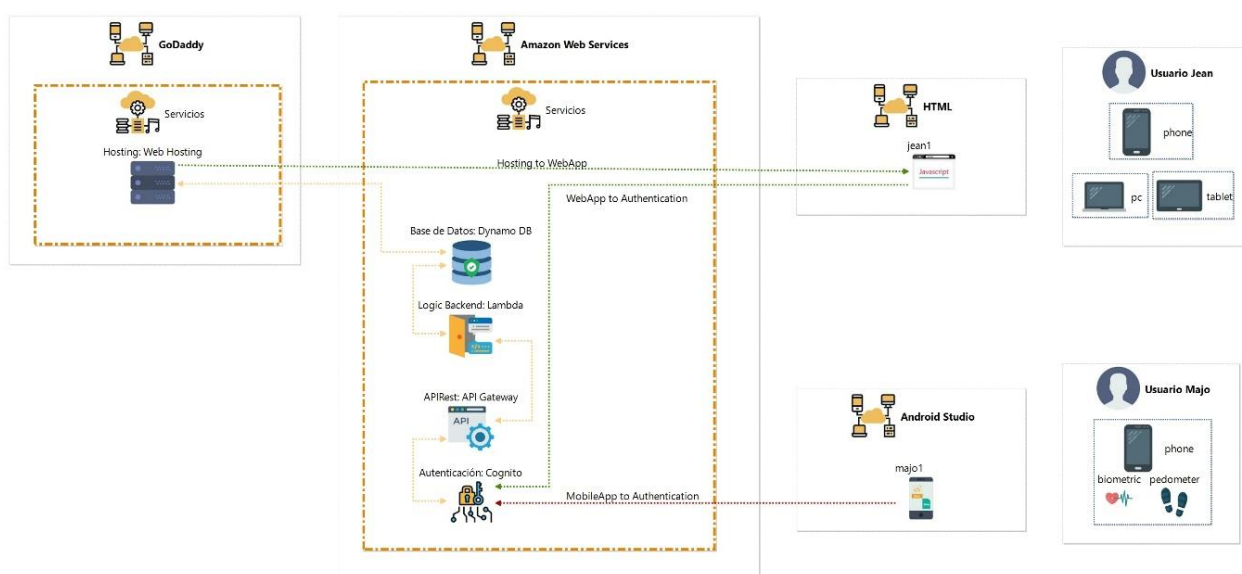


Figura 59 DSL de escenario de prueba de arquitectura

Se realizó el testeo de la arquitectura, mediante el servicio de hosting de la página web con un proveedor distinto de AWS, GoDaddy, modificando la URL de origen en la seguridad CORS en la API RESTful, se verificó la funcionalidad del sistema sin alterar su resultado final integrado con los demás servicios y se demostró que la arquitectura es válida para aplicaciones con diferentes proveedores sin alterar el desempeño de la aplicación.

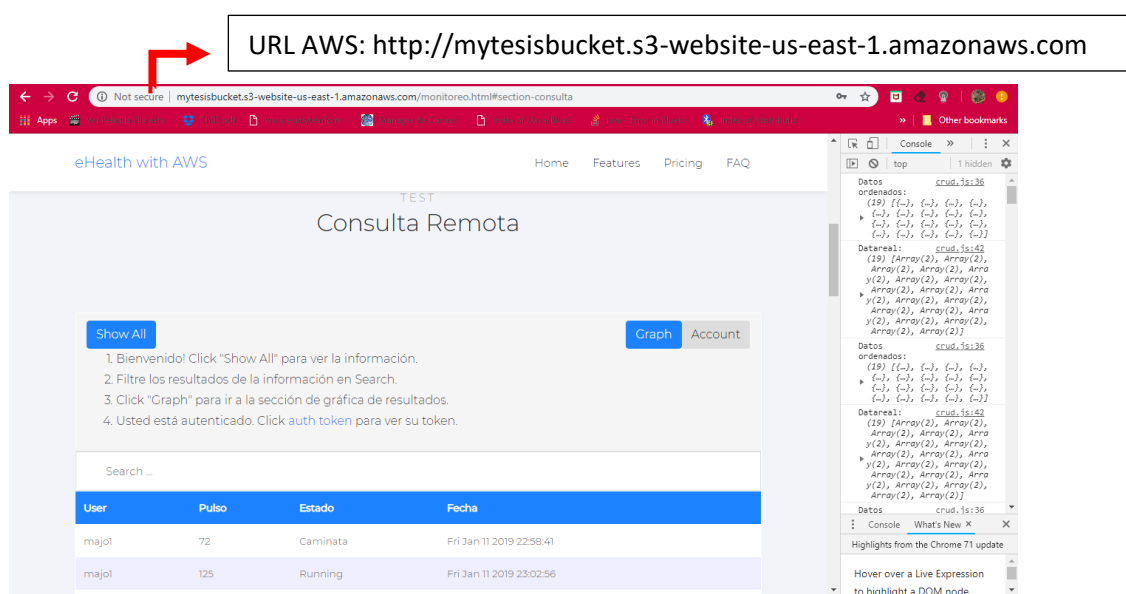


Figura 60 Prueba de arquitectura con Web hosting AWS



Figura 61 Prueba de arquitectura con Web hosting GoDaddy

5.2. PRUEBAS DE RENDIMIENTO

Para las pruebas de rendimiento del lado del servidor se realizó un testeo de la API RESTful desarrollada para la página web en el software JMeter, se realizaron tres escenarios de prueba de carga simulando 500, 1000 y 1500 usuarios concurrentes realizando peticiones HTTP GET al servidor durante un período de 60 segundos, la finalidad de este test fue determinar la escalabilidad que proporciona la aplicación desarrollada con AWS.



Figura 62 Prueba de carga con 500 usuarios concurrentes durante 60 segundos

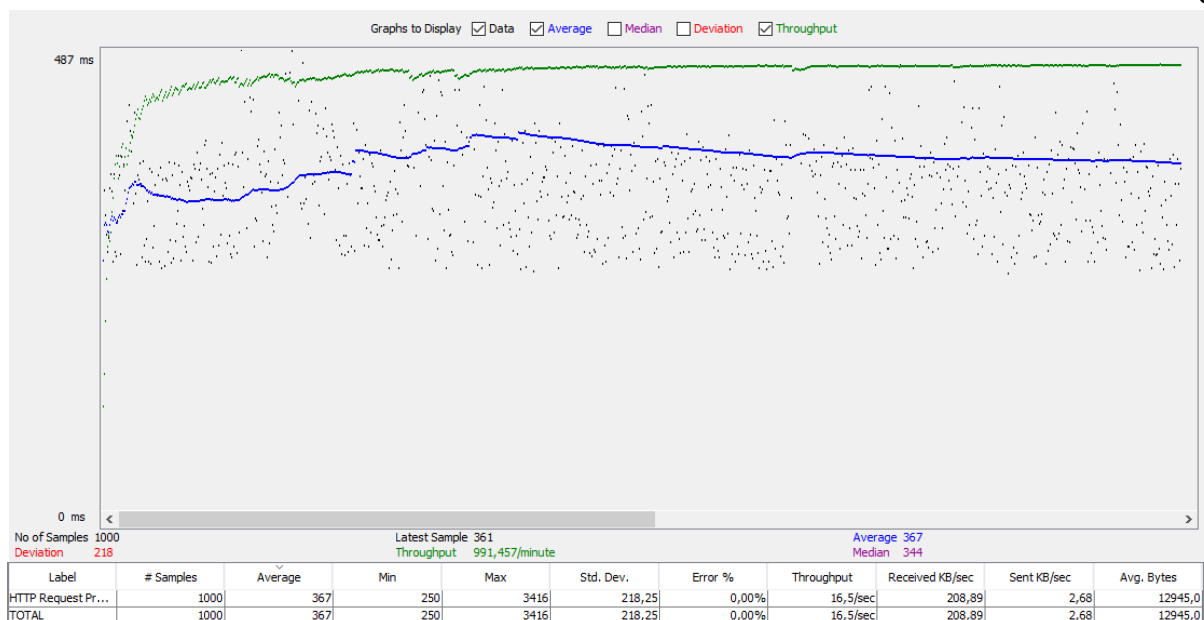


Figura 63 Prueba de carga con 1000 usuarios concurrentes durante 60 segundos

En los dos primeros escenarios de prueba, Figura 62 y Figura 63, se verificó que todas las solicitudes fueron realizadas con éxito con un porcentaje de error de 0% y el promedio del tiempo de cada solicitud se realizó en 316 ms para la carga de 500 usuarios y 367 ms para 1000 usuarios.

En el caso del tercer escenario se verificó un porcentaje de error de 0.07% en las solicitudes HTTP GET realizadas, con un tiempo promedio de 1028 ms por cada solicitud.

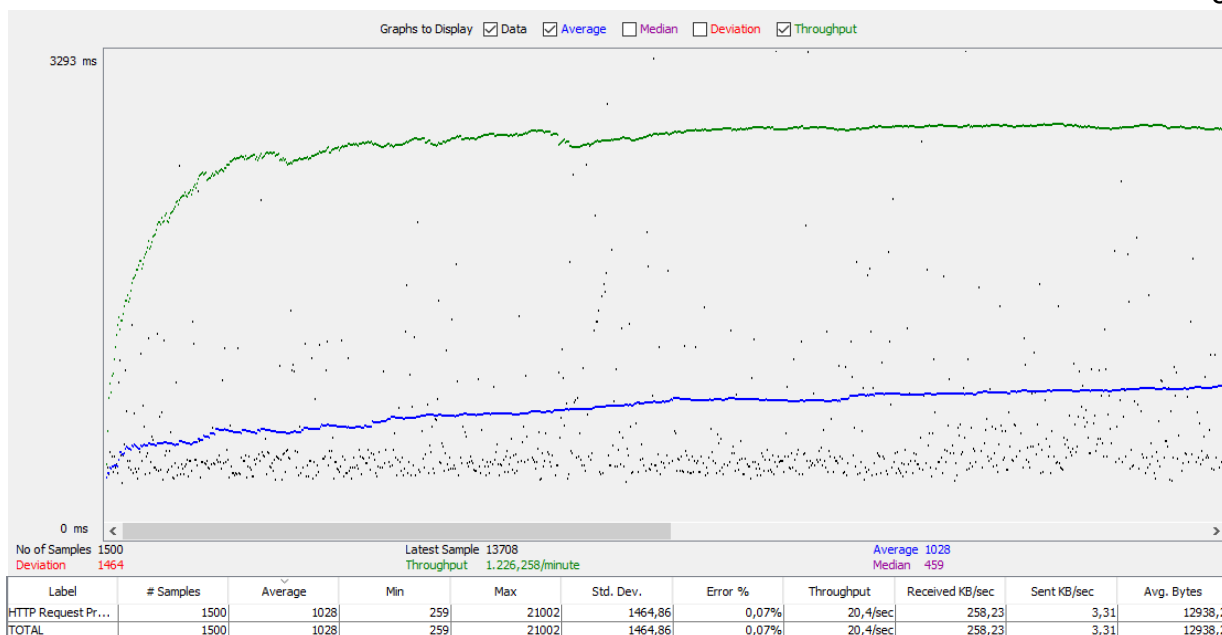


Figura 64 Prueba de carga con 1500 usuarios concurrentes durante 60 segundos

Para las pruebas de rendimiento del lado del cliente se realizó el testeo del tráfico de la aplicación móvil en Android Profiler, para verificar su comportamiento en la red cuando se realizan las peticiones HTTP POST al servidor.

En el escenario de peticiones HTTP POST de autenticación al servidor, se verificó el éxito de conexión de cada una de las peticiones en un tiempo promedio de 378 ms y un estado 200 sin fallo.

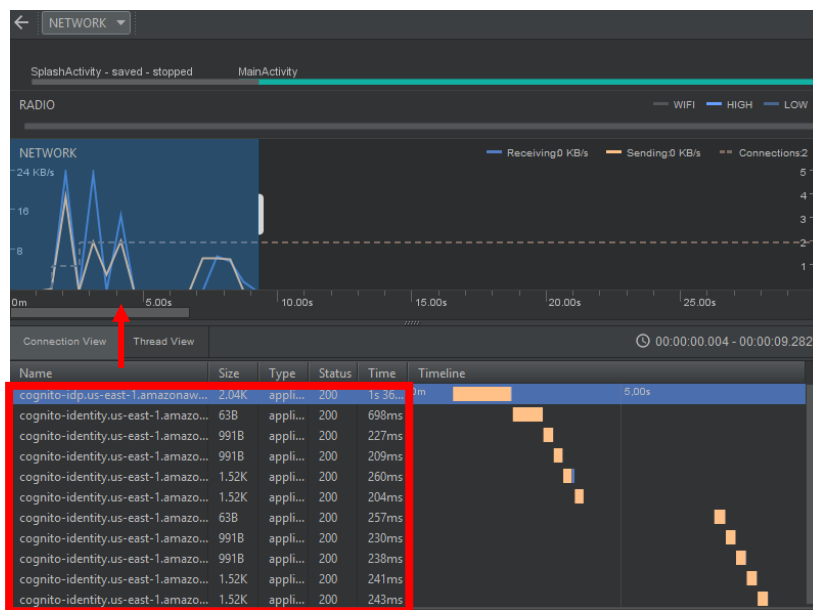


Figura 65 Peticiones HTTP POST a Amazon Cognito

En el escenario donde el usuario realiza una petición HTTP POST al servidor en la etapa de almacenar datos en DynamoDB, realizando tres veces la operación de creación de un objeto en la tabla, en el servidor se verificó el éxito de conexión de la petición en un tiempo promedio de 600 ms y un estado 200 sin fallo.

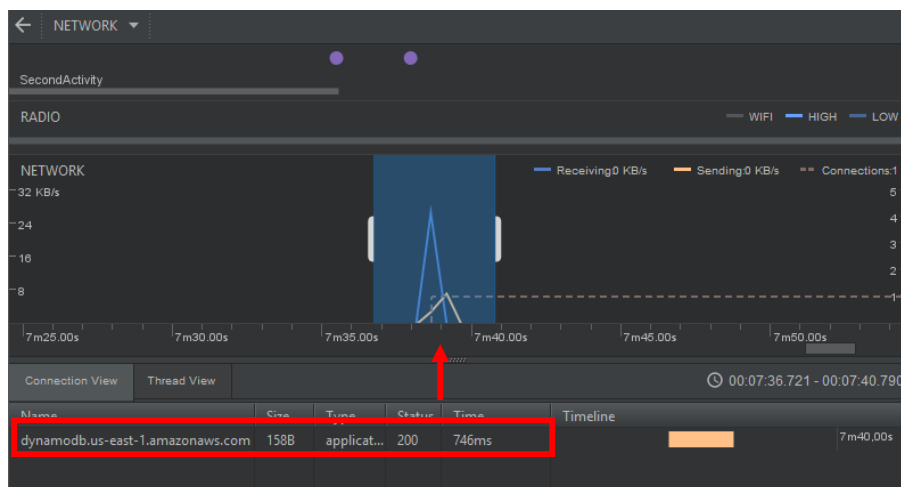


Figura 66 Peticiones HTTP POST a Amazon DynamoDB

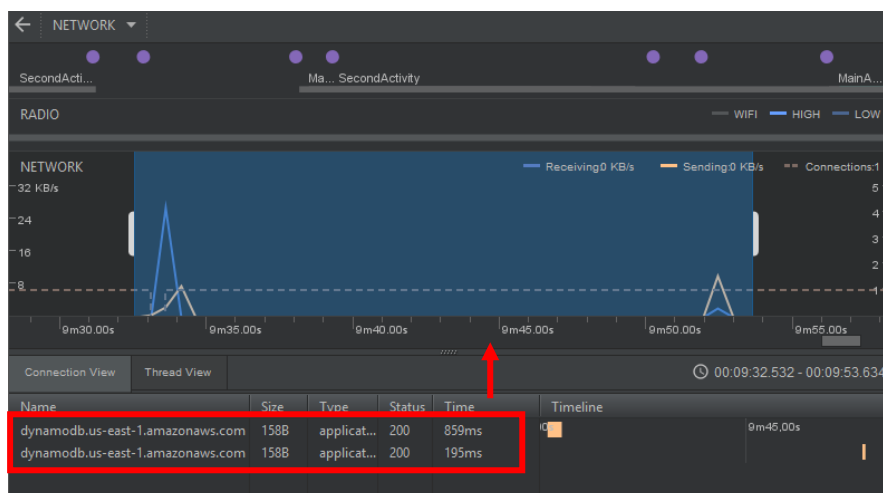


Figura 67 Peticiones HTTP POST a Amazon DynamoDB

5.3. PRUEBAS DE FUNCIONAMIENTO

Para la validación del funcionamiento de la aplicación desarrollada se realizó la autenticación de un usuario, la medición del ritmo cardíaco y los pasos. Se verificó que los datos se hayan guardado correctamente y que en la consulta remota desde la página web se reflejen los datos correctamente. El anexo B verifica el proceso.

En cuanto a los pulsos medidos con la aplicación se realizó una comparación con las lecturas de la aplicación Samsung Health de Samsung, que mide el ritmo cardíaco con su propio sensor biométrico HRM y al que no se pudo tener acceso para el trabajo. Validando así el algoritmo de (Zulhilmi Bin , 2013) que se utilizó.

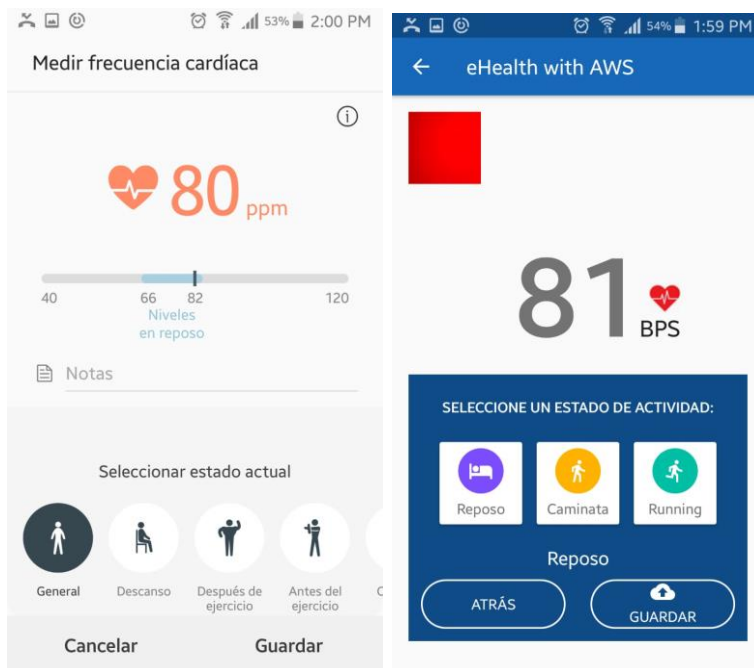


Figura 68 Verificación frecuencia cardíaca (a)

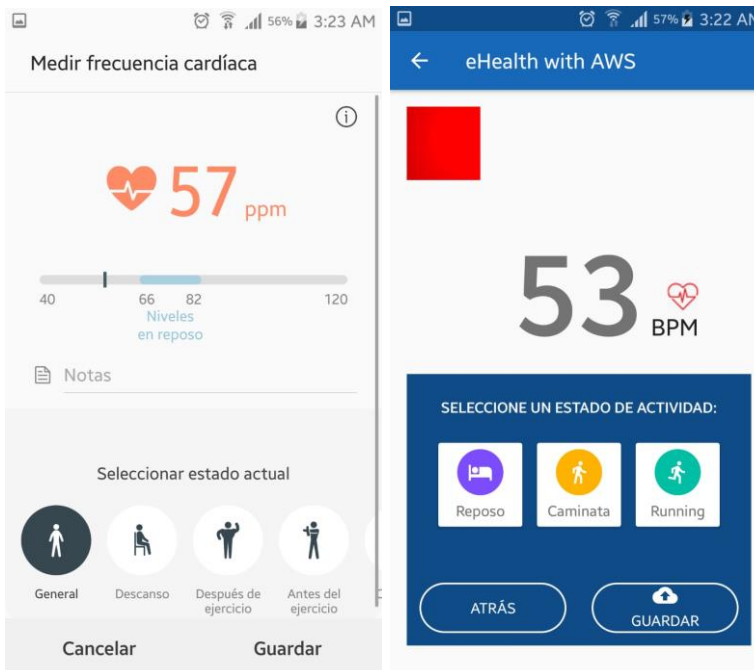


Figura 69 Verificación frecuencia cardíaca (b)

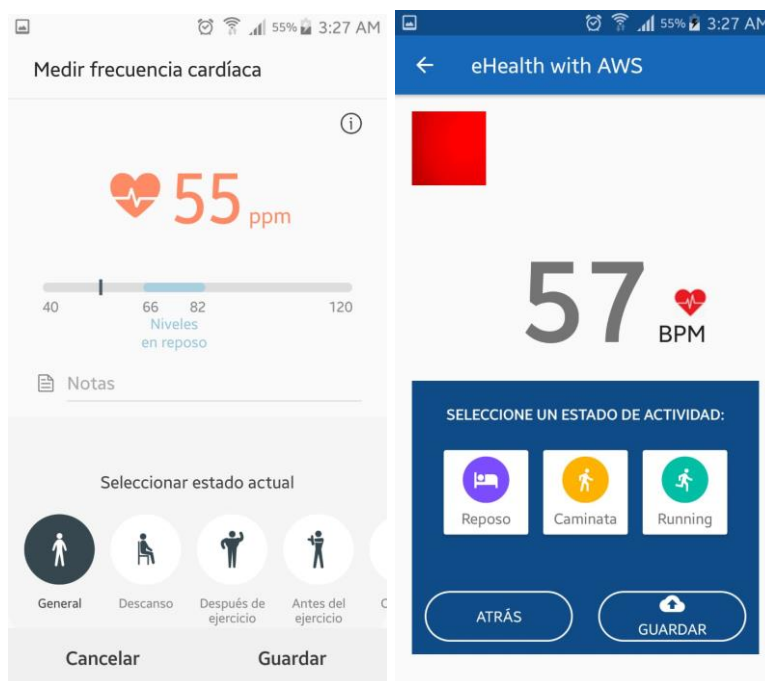


Figura 70 Verificación frecuencia cardíaca (c)

En las figuras 68, 69 y 70 se comprueba que las lecturas del ritmo cardíaco son reales, y entre las lecturas realizadas, el mayor porcentaje de error es el de la Figura 69, con un porcentaje de 7.02%. Y el menor error es el correspondiente a la Figura 68, con un porcentaje de 1.25%. Las lecturas se realizaron en estado de relajación y a temperatura ambiente pues cuando existe actividad intensa, la actividad cardíaca es muy variable, siendo imposible la comparación.

CAPÍTULO 6: CONCLUSIONES Y RECOMENDACIONES

5.1. CONCLUSIONES

- Se realizó un estudio sobre los sistemas cloud de eHealth. Estos sistemas actualmente tienen un gran potencial y de hecho se han elaborado arquitecturas o casos específicos donde combinan el ámbito de la salud y el deporte con las TICs. El internet ha sido su mayor aliado para desarrollar la asistencia remota sin embargo su punto a considerar es la seguridad que le dan a sus aplicaciones y la diversidad de equipos de Telemedicina y su integración.
- Se realizó el estudio del modelado como herramienta para la elaboración de arquitecturas y su integración con los modelos de servicio cloud, donde se destaca que el modelo de Infraestructura como servicio brinda mejores ventajas para crear aplicaciones fácilmente escalables en la nube. Sin embargo el utilizar plataformas IaaS para investigación puede resultar limitante si no se cuenta con el recurso monetario para mantener los servicios activos. Amazon Web Services es una plataforma IaaS muy completa, pero incluso la cuenta con un año gratis de prueba

tiene límites de consumo convirtiendo a AWS en un proveedor poco viable para el ámbito de la investigación y aplicaciones a pequeña escala.

- Se diseñó una arquitectura genérica multiplataforma para una familia de aplicaciones cloud en el dominio de eHealth mediante el uso de ingeniería basada en modelos. Al utilizar las técnicas de MDA y MDE en la arquitectura propuesta, se logró diseñar un metamodelo cuya representación en DSL proporciona una automatización de procesos, que permite a desarrolladores estructurar modelos de aplicación en el dominio seleccionado.
- Se desarrolló e implementó un sistema eHealth basado en la arquitectura diseñada, el metamodelo propuesto facilitó la definición de los requerimientos de hardware y software para su integración y ejecución con servicios en la nube. Con el uso de tecnologías para el desarrollo de aplicaciones móviles y web se obtuvo un sistema funcional y visualmente intuitivo para el usuario.
- Al utilizar MDA para desarrollar la arquitectura, se logró simplificar las decisiones técnicas para la implementación en el caso de uso, pues en el metamodelo se definieron los principios del dominio del problema y la implementación se encargó de decisiones sobre la tecnología y plataformas a utilizar.
- El sistema eHealth desarrollado utilizó el protocolo IoT HTTP (REST/JSON), donde el nodo IoT es representado por el teléfono móvil y sus sensores: biométrico y podómetro. El nodo tomó el rol de cliente cuando realizó peticiones de registro y autenticación a AWS y como servidor cuando proporcionó los valores de los sensores al servidor AWS. Debido a la característica de definición de roles que provee el protocolo HTTP, aporta simplicidad al sistema.

- Se implementó mecanismos de seguridad para la aplicación móvil y web. La aplicación móvil tiene acceso a la base de datos mediante las credenciales válidas de permiso que se obtienen en el proceso de autenticación con AWS, en cuanto a la aplicación web el acceso a la base de datos es mediante un token de autorización, que otorga las credenciales de permisos a la API Restful que realiza peticiones HTTP con seguridad TLS al servidor, adicionalmente se le agregó una política de CORS a su invocación, restringiendo dominios y brindando transferencia de datos segura entre el navegador y el servidor web.
- Una herramienta multiplataforma en sistemas IoT integra dispositivos heterogéneos y en las pruebas de arquitectura realizadas se verificó que el metamodelo planteado es multiplataforma y orientado a microservicios, pues se integró la misma aplicación cliente con otro proveedor de hosting GoDaddy y su funcionamiento con los servicios restantes de AWS no se vieron alterados.
- AWS como plataforma IaaS seleccionada demostró una alta escalabilidad en el caso de uso desarrollado, en las pruebas de rendimiento que se ejecutaron se obtuvo que los servicios utilizados son altamente escalables y las pruebas de carga de usuarios demostraron que las peticiones HTTP realizadas se atienden cumpliendo los SLA (Acuerdos de nivel de servicio) que AWS garantiza a sus clientes, sin embargo el punto a considerar son los costos que pueden generarse, si la cuenta se utiliza solamente para el desarrollo de aplicaciones con fines investigativos.
- Los sensores HRM (Heart Rate Monitor) vienen incluidos en dispositivos Samsung de alta gama con API mayor a 21, utilizan como fuente una luz LED óptica y un

sensor de luz LED, miden los pulsos con la cantidad de luz reflejada en el dedo de la persona. Como alternativa de solución para la implementación se utilizó el algoritmo para sensado de ritmo cardíaco desarrollado por (Zulhilmi Bin , 2013) que realiza procesamiento digital de la imagen mediante la cámara del celular y el flash. De acuerdo a las pruebas realizadas, sus mediciones validan al sistema frente al sensor HRM. Es necesario acotar que este tipo de sensado de ritmo cardíaco se dirige a personas que no han presentado síntomas de enfermedades cardíacas, denominadas personas con prevención primaria según la OMS.

- Se utilizó el flujo de inscripción/inicio de sesión para la configuración del backend móvil de AWS con la facilidad de la herramienta de Mobile Hub, debido a que la misma crea automáticamente un grupo de usuarios de Amazon Cognito, al cual se le aplican las reglas de contraseña que se propuso, garantizando la conexión correcta de la aplicación móvil al servicio. Mediante la clase handler de java se gestiona la respuesta y si las credenciales proporcionadas por AWS son iguales se da la transición a la actividad principal de la aplicación. El mismo flujo de inscripción/inicio de sesión se utilizó para la aplicación web, mediante Java Script.
- Se desarrolló la aplicación móvil en Android Studio y se obtuvo una solución que cumplió con los requisitos previstos y debido a la amplia documentación que brinda AWS se integró los servicios de autenticación y base de datos a través de las clases de Java y los archivos JSON de configuración.

5.2. RECOMENDACIONES

- En cuanto al desarrollo del DSL, se recomienda agregar el metamodelo al proyecto antes de agregar los nodos, contenedores y enlaces a la representación, en ocasiones no generan errores pero otras veces bloquea la funcionalidad del DSL.
- Se recomienda desarrollar la página web localmente antes de subirla al bucket de Amazon S3, y si se realizan cambios es importante actualizar únicamente los archivos que se han ido modificando, esto debido al límite de la capa gratuita en cuanto a solicitudes POST y DELETE que ofrece el servidor.
- Se recomienda trabajar la aplicación móvil con los SDKs oficiales de los sensores debido a la existencia de documentación y que su funcionamiento es previamente testeado y aceptado por estándares universales, en cuanto al ritmo cardíaco para que sus lecturas sean más exactas y precisas, y tenga un respaldo mayor. Sin embargo para aplicaciones de medicina, en personas con riesgo cardiovascular alto, se debe optar por sensores biomédicos sofisticados y dedicados.

5.3. TRABAJOS FUTUROS

- A partir del metamodelo de la arquitectura planteada, que es un modelo PIM, se propone profundizar en el ámbito de las transformaciones de modelos MDA, con

la finalidad elevar el nivel de interoperabilidad entre el PIM y sus PSM, llegando a construir fragmentos de código para que la implementación de las aplicaciones se automatice y los tiempos de desarrollo disminuyan.

- Se propone utilizar el metamodelo desarrollado como base para generar una nueva arquitectura que integre más dominios de IoT y cloud computing con la finalidad de estructurar una arquitectura unificada para una diversidad mucho más amplia de sistemas y casos de uso.
- En cuanto a la aplicación elaborada se propone desarrollar la parte web, agregando una funcionalidad que permita tener control y validación sobre las personas que se registran, con la finalidad de que pueda ser un sistema más controlado dirigido a instituciones de salud o entrenamiento.
- En lo referente a la aplicación móvil se puede implementar un servicio de configuración de perfil del usuario, con el fin de poder cargar mayores datos al servidor, agregar un sistema de Machine Learning, y generar notificaciones inteligentes al lado del monitoreo remoto.
- Se propone realizar la implementación del sistema con plataformas de servicios diferente a Amazon Web Services y otro protocolo IoT, replicando su funcionalidad, y realizar una comparación.

BIBLIOGRAFÍA

- Garai, Á., Attila, A., & Péntek, I. (2016). Cognitive Telemedicine IoT Technology for Dynamically Adaptive eHealth Content Management Reference Framework embedded in Cloud Architecture. *IEEE International Conference on Cognitive Infocommunications*.
- Rivas Costa, C., Anido Rifón, L., & Fernández Iglesias, M. (2016). An Open Architecture to Support Social and Health Services in a Smart TV Environment. *IEEE Journal of Biomedical and Health Informatics*.
- Android. (2018). *Android Developers*. Obtenido de <https://developer.android.com/studio/intro/?hl=es-419>
- AWS. (2018). *Amazon Web Services*. Obtenido de https://aws.amazon.com/choosing-a-cloud-platform/?nc2=h_l2_cc
- AWS. (2018). *Capa gratuita de AWS*. Obtenido de <https://aws.amazon.com/es/free/?awsf.Free%20Tier%20Types=categories%2312monthsfree>
- Bootstrap. (2018). *Bootstrap*. Obtenido de <https://getbootstrap.com/>
- Campoverde, A., Mazón, B., & Hernández, D. (2015). Cloud computing con herramientas open-source para Internet de las cosas. *Maskana*.
- Castillejo Parrilla, P. (2015). Contribution towards intelligent service management in wearable and ubiquitous devices. Obtenido de http://oa.upm.es/37254/1/PEDRO_CASTILLEJO_PARRILLA.pdf

- Catarinucci, L., De Donno, D., Mainetti, L., Palano, L., Patrono, L., Stefanizzi, M. L., & Tarricone, L. (2015). An IoT-Aware Architecture for Smart Healthcare Systems. *IEEE Internet of Things Journal*, 2(6), 515 - 526.
- Eclipse. (2018). *Eclipse Modeling Framework (EMF)*. Obtenido de <https://www.eclipse.org/modeling/emf/>
- Eclipse. (2018). *Sirius*. Obtenido de <https://www.eclipse.org/sirius/overview.html>
- Espiniza, C., Pérez, M. J., & Peralta, M. (2017). El Internet de las cosas: Antecedentes, conceptualización y riesgos. *UTMACH Conference Proceedings*, 1(ISSN2588-056X).
- Evans, D. (2011). *Internet of Things, la próxima evolución de Internet lo está cambiando todo*. Técnico, Cisco, Grupo de Soluciones Empresariales para Internet (IBSG). Obtenido de https://www.cisco.com/c/dam/global/es_es/assets/executives/pdf/Internet_of_Things_IoT_IBSG_0411FINAL.pdf
- Gaitán Peña, C. A. (2017). Líneas de Productos Software: Generando Código a Partir de Modelos y Patrones. *Scientia et Technica Año XXII*, 22(2).
- García, J., García, F., Palechano, V., Vallecillo, A., Vara, J., & Chicote, C. (2012). *Desarrollo de Software Dirigido por Modelos*. Madrid: RA-MA Editorial.
- González García, A. (junio de 2017). *IoT: Dispositivos, tecnologías de transporte y aplicaciones*. Obtenido de <http://openaccess.uoc.edu/webapps/o2/bitstream/10609/64286/3/agonzalezgarcia0TFM0617memoria.pdf>

- Gope, P., & Hwang, T. (2015). BSN-Care: A Secure IoT-based Modern Healthcare System Using Body Sensor Network. *IEEE Sensors Journal* , 16(5), 1368 - 1376.
- GPC. (2017). *Amazon Web Services AWS – Infografía Estructura Productos*. Obtenido de https://gpcinc.mx/blog/que-es-saas-paas-iaas/amazon-web-services-infografia-estructura-productos_3-copia-9/
- Gubbi, J., Buyya , R., Marusic, S., & Palaniswami, M. (2013). Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions. *Future Generation Computer Systems*, 1645–1660.
- IBM. (2015). *IaaS PaaS SaaS - Modelos de servicio cloud*. Obtenido de <https://www.ibm.com/cloud-computing/es-es/learn-more/iaas-paas-saas/>
- Kardoš, M., & Drozdová, M. (2010). Analytical Method of CIM to PIM Transformation in Model Driven Architecture (MDA). *JIOS*, 34(1), 89-99.
- Kulkarni, A., & Sathe, S. (2014). Healthcare applications of the Internet of Things: A Review. *International Journal of Computer Science and Information Technologies*, 5, 6229-6232.
- Mistry, A. (2018). *Expert AWS development* (Primera ed.). Birmingham, UK: Packt Publishing.
- Moguel Márquez, J. E. (2018). *Una arquitectura orientada a servicios y dirigida por eventos para el control inteligente de UAVs multipropósito*. Tesis Doctoral, Universidad de Extremadura.
- Mora, B., Ruiz, F., García, F., & Piattini, M. (2006). *Definición de Lenguajes de Modelos MDA vs DSL*. Universidad de Castilla-La Mancha, Departamento de Tecnologías y Sistemas de Información.

Morales, M. (s.f.). *Internet of things (IoT) en la transformación digital de las empresas.*

Íncipy, Data Intelligence. Obtenido de <http://www.fundacionseres.org/Lists/Informes/Attachments/987/150923%20internet-of-things.pdf>

Murazzo, M., & Rodríguez, N. (2010). Mobile Cloud Computing. *XII Workshop de Investigadores en Ciencias de la Computación*, 522-526.

Nahuel Delía, L. (2017). *Desarrollo de Aplicaciones Móviles Multiplataforma*. Buenos Aires.

OMS. (2008). *Prevención de las enfermedades cardiovasculares, Guía de bolsillo para la estimación y el manejo del riesgo cardiovascular*. Ginebra: Ediciones de la OMS.

Ortiz Bonilla, F. (2017). *Monitorización del pulso de usuario mediante la lectura de sensores Bluetooth LE usando Eclipse Kura y AWS IoT*. Universidad de Sevilla, Departamento de Ingeniería Telemática, Sevilla.

Palao Cruz, C. (2017). *Desarrollo de un sistema IoT integrado con dispositivos eHealth para la detección automática de la variabilidad cardíaca*. Tesis, Universidad Politécnica de Valencia, Escuela Técnica Superior de Ingenieros de Telecomunicación, Valencia.

Romero, J. (marzo de 2017). *iTechFit. Todo sobre la Tecnología aplicada al Deporte y la Salud*. Obtenido de <https://bitnarios.com/itechfit-la-tecnologia-aplicada-al-deporte-la-salud/>

Samsung. (2018). *Samsung Developers*. Obtenido de <https://developer.samsung.com/galaxy/sensor-extension>

Sebestyen, G., Hangan, A., Oniga, S., & Gál, Z. (2014). eHealth Solutions in the Context of Internet of Things. *IEEE International Conference on Automation, Quality and Testing, Robotics*.

Steinberg, D., Budinsky, F., Merks, E., & Paternostro, M. (2008). *EMF: Eclipse Modeling Framework*.

TechTarget. (2016). *A comparison of Azure, AWS, and Google cloud services*. Obtenido de

https://cdn.ttgtmedia.com/searchCloudComputing/downloads/A_comparison_of_Azure_AWS__and_Google_cloud_services.pdf

Zulhilmi Bin , M. R. (2013). *Realtime Health Monitoring based on Android Platform*. Universiti Teknologi Malaysia.