



**UNIVERSIDAD
NACIONAL DE
INGENIERÍA**

*Diseño e implementación del core level de una
plataforma transversal basado en arquitecturas
Fog Computing*

Moreno Vera, Felipe Adrian

Faculty of Science

Asesor:

Mag. Castillo Cara, Jose Manuel

Jurados:

Mag. Tenorio Trigos, Alonso

Mag. Nuñez Medrano, Yuri

Tesis presentada a

Escuela Profesional de Ciencia de la Computación

como parte de los requisitos para obtener el grado de

Bachiller en Ciencias con mención en Ciencia de la Computación.

Universidad Nacional de Ingeniería

Julio, 2017 - Lima - Perú

*Este trabajo esta dedicado a mis padres,
hermanos y demás familiares que siempre
han estado apoyando y depositando su
confianza en mi, siempre estarán en mi
mente y mi corazón.*

Resumen

En este trabajo se presenta una arquitectura basado en Fog Computing para poder minimizar costo computacional utilizado generalmente en los servidores Cloud centralizados, a su vez, generar sistemas de filtrado y pre-procesado de información en los servidores (o también llamados nodos) Fog aligerando el envío y cantidad de información entre la capa final (Edge) hacia el Cloud.

Keywords: Fog Computing, Cloud Computing, REST services, Servers, Databases, Web services, JSON formats.

Prefacio

Una de las principales necesidades que se presentan al momento de seguir escalando a nivel de cómputo, a nivel de almacenamiento, a nivel de procesamiento, etc. es como controlar la información que se genera y todo el proceso que tiene que realizar para llegar a un almacenado o visualización o procesado correcto.

Actualmente la ausencia de herramientas y arquitecturas que nos indiquen maneras para optimizar y reducir la cantidad de consumo de energías y poder de cómputo produce que tecnologías como Big Data o Data Science tenga un alto grado de dificultad en su implementación.

Es por ello que se propone la implementación y desarrollo de una arquitectura basada en tecnologías de *Fog Computing* la cual reduce y optimiza los procesos mencionados anteriormente, haciendo un óptimo uso de los recursos computacionales con que se cuentan actualmente a través de metodologías de desarrollo ágil.

Agradecimientos

A la **Universidad Nacional de Ingeniería** y al **Centro de Tecnologías de Información y Comunicaciones**, por brindarme un espacio tan variado en conocimiento y personas con las cuales interactuar y poder plantear, desarrollar e implementar el presente trabajo.

Siglas y Abreviaturas

CMM *Capability Maturity Model*

AMQP *Advanced Message Queuing Protocol*

API *Application Programming Interface*

CoAP *Constrained Application Protocol*

CSS *Cascading Style Sheets*

DBMS *Data Base Management System*

HTML *HyperText Markup Language*

HTTP *Hypertext Transfer Protocol*

JSON *JavaScript Object Notation*

MQTT *Message Queue Telemetry Transport*

MVC *Model View Controller*

MVW *Model View Whatever*

M2M *Machine To Machine*

P2P *Peer To Peer*

QoS *Quality of Service*

RAD *Rapid Application Development*

REST *REpresentational State Transfer*

SMTP *Simple Mail Transfer Protocol*

SOAP *Simple Object Access Protocol*

SPA *Single-Page Application*

SSH *Secure Shell*

SSL *Secure Sockets Layer*

TLS *Transport Layer Security*

Índice general

Resumen	V
Índice de figuras	XV
Índice de tablas	XVI
1. Introducción	1
1.1. Motivación	1
1.2. Objetivos	2
1.3. Estructura del Seminario	3
2. Estado del Arte	5
2.1. Introducción	5
2.2. Monitoreo Remoto de un Sistema	6
2.2.1. Estado Actual	6
2.3. Aplicaciones de censado en Tiempo Real	7
2.3.1. ClearBlade	7
2.3.2. Hologram	8
2.4. Conclusiones	9
3. Tecnologías, Herramientas y Metodologías Utilizadas	11
3.1. Metodologías de Desarrollo	11
3.2. Tecnologías y Herramientas de Desarrollo	12
3.2.1. Software Controlador de Versiones	12
3.2.2. Sistema de Gestor de Base de datos	12
MongoDB	12
3.2.3. Go language	13

3.2.4.	Javascript	13
3.2.5.	HTML5, CSS3 y Bootstrap	13
	HTML5	13
	CSS3	14
	Bootstrap	14
3.2.6.	Google Maps API	14
3.2.7.	AngularJS	14
3.2.8.	Protocolos de comunicación	15
	Simple Mail Transfer Protocol	15
3.2.9.	Protocolos <i>IoT</i>	15
	HyperText Transfer Protocol	16
	Message Queue Telemetry Transport	16
	Constrained Application Protocol	16
	Advanced Message Queuing Protocol	16
3.2.10.	Patrones de Arquitectura	18
	MVC	18
	MVW	20
4.	Análisis y Diseño de la Plataforma	22
4.1.	Análisis de la Plataforma	22
4.1.1.	Actores	22
4.1.2.	Reglas de Negocio	23
4.1.3.	Requisitos de la Plataforma	24
	Historias de Usuario Finales	24
4.1.4.	Modelo de Datos	26
4.2.	Diseño de la Plataforma	26
4.2.1.	Estructura de la base de datos	27
4.2.2.	Diseño de la arquitectura de servicios	28
5.	Caso de Estudio	34
5.1.	Inicio de sesión	34
5.2.	Mi Cuenta	36

5.3. Administración de usuarios	36
5.4. Administración de Módulos	38
5.5. Información	39
5.6. Graficas	40
5.7. Mapas	42
6. Conclusiones y Trabajo Futuro	44
6.1. Conclusiones	44
6.2. Trabajo Futuro	45
6.3. Comentario Final	46
Bibliografía	52
A. Definiciones	54
A.1. Servicios Web	54
A.2. API Web	57
A.3. Recursos Web	59
A.4. Seguridad en Redes	62
B. Ingeniería del Software	63
B.1. Casos de Uso	63
B.1.1. Administrador	63
B.1.2. Usuario	64
B.2. Especificaciones de los Casos de Uso	64
B.2.1. Caso de uso: Usuario	64
B.2.2. Caso de uso: Administrador	66
C. Desarrollo de Software Ágil	68
C.1. SCRUM	68
C.1.1. Componentes	69
C.1.2. Proceso	70
C.2. Sprint Backlogs	72
C.2.1. Primer Sprint Backlog	72

C.2.2. Segundo Sprint Backlog	73
C.2.3. Tercer Sprint Backlog	74
C.2.4. Cuarto Sprint Backlog	75
C.2.5. Quinto Sprint Backlog	76
C.2.6. Sexto Sprint Backlog	76
C.2.7. Séptimo Sprint Backlog	77
C.2.8. Octavo Sprint Backlog	78
C.2.9. Noveno Sprint Backlog	78
C.2.10. Décimo Sprint Backlog	79
C.2.11. Decimo Primer Sprint Backlog	80
C.2.12. Decimo Segundo Sprint Backlog	80
Tiempo de desarrollo total de la plataforma	81
D. Manual y Guías de instalación	82
D.1. Servidor Cloud	82
D.1.1. Lenguaje de programación Go	82
BeeGo	83
GoMail	83
Paho MQTT Golang	83
Mgo	83
D.2. Servidores Fog	84
D.2.1. Lenguaje de programación Python	84
PIP	84
Paho MQTT Python	84
PyMongo	84
Coloredlogs	85

Índice de figuras

2.1. ClearBlade dashboard, sistema de monitorización de los dispositivo asociados. Fuente: ClearBlade Website. [1]	8
2.2. Hologram dashboard, software destinado al monitoreo en tiempo real. Fuente: Hologram Website. [2]	9
3.1. Patrón Model-View-Controller. Fuente: Wikipedia MVC. [3]	20
3.2. Patrón Model-View-Whatever. Fuente: AngularJS, a MVW Framework. [4]	21
4.1. Diagrama Entidad-Relación. Fuente: Elaboración propia.	27
4.2. Diagrama de Clases. Fuente: Elaboración propia.	28
4.3. Representación de servidor Cloud que realiza un servicio web basado en REST. Fuente: Elaboración propia.	29
4.4. Representación de un servidor Cloud que funciona como Broker MQTT. Fuente: Elaboración propia.	30
4.5. Representación de un servidor Fog que funciona como Broker MQTT. Fuente: Elaboración propia.	31
4.6. Representación de un servidor Cloud y diversos servidores Fog, los cuales pueden comunicarse mediante SSH o MQTT. Fuente: Elaboración propia.	32
4.7. Representación de la arquitectura Fog Computing de la plataforma. Fuente: Elaboración propia.	33
5.1. Inicio de Sesión. Fuente: Elaboración propia.	34
5.2. Home del administrador. Fuente: Elaboración propia.	35
5.3. Home de un usuario. Fuente: Elaboración propia.	35
5.4. Crear Usuarios. Fuente: Elaboración propia.	37

5.5. Eliminar Usuarios. Fuente: Elaboración propia.	37
5.6. Crear Módulos. Fuente: Elaboración propia.	38
5.7. Eliminar Módulos. Fuente: Elaboración propia.	39
5.8. Información de los Usuarios. Fuente: Elaboración propia.	39
5.9. Información de los Módulos. Fuente: Elaboración propia.	40
5.10. Información de los Sensores. Fuente: Elaboración propia.	40
5.11. Visualización seleccionando por Modulo. Fuente: Elaboración propia.	41
5.12. Visualización seleccionando por tipo sensor. Fuente: Elaboración propia.	41
5.13. Historico de datos. Fuente: Elaboración propia.	42
5.14. Ubicacion de los módulos junto con el área que cubren. Fuente: Elaboración propia.	43
5.15. Mapas de calor de los valores censados representados en las áreas de los módulos. Fuente: Elaboración propia.	43
A.1. Arquitectura de un Servicio Web basado en SOAP. Fuente: Elaboración propia.	55
A.2. Estructura de un mensaje SOAP entre el cliente web y el servicio rest. Fuente: Elaboración propia.	56
A.3. Arquitectura de un Servicio Web basado en REST. Fuente: Elaboración propia.	57
A.4. Ejemplo de restful routing. Fuente: Elaboración propia.	58
A.5. Estructura de un URI. Fuente: Elaboración propia.	59
B.1. Casos de uso del administrador del sistema. Fuente: Elaboración propia.	63
B.2. Casos de uso del usuario. Fuente: Elaboración propia.	64
C.1. Estrategia de desarrollo ágil SCRUM. Fuente: Software Testing Material. [5]	71

Índice de cuadros

3.1. Tabla comparativa de características de protocolos de mensajería para sistemas IoT: MQTT, CoAP, AMQP y HTTP. Fuente: [6] . . .	18
B.1. Especificación de los Casos de uso del Usuario - Solicitar módulo	65
B.2. Especificación de los Casos de Uso del Usuario - Cancelar Servicio	65
B.3. Especificación de los Casos de Uso del Usuario - Consultar Data .	66
B.4. Especificación de los Casos de Uso del Administrador	67
C.1. Primer Sprint Backlog	72
C.2. Segundo Sprint Backlog	73
C.3. Tercer Sprint Backlog	74
C.4. Cuarto Sprint Backlog	75
C.5. Quinto Sprint Backlog	76
C.6. Sexto Sprint Backlog	76
C.7. Séptimo Sprint Backlog	77
C.8. Octavo Sprint Backlog	78
C.9. Noveno Sprint Backlog	79
C.10.Décimo Sprint Backlog	79
C.11.Decimo Primer Sprint Backlog	80
C.12.Decimo Segundo Sprint Backlog	80

Capítulo 1

Introducción

En este capítulo se presenta la motivación por el cual se decidió esta temática como Seminario de Tesis, basado en las tecnologías de actuales, de alto impacto y mayor requerimiento en la actualidad. Además de la estructura del presente trabajo.

1.1. Motivación

Las ciudades más pobladas alrededor de todo el mundo, sin importar que nivel o calidad de vida tengan, tienen un alto índice de contaminación, Kabul - Afghanistan y Accra - Ghana poseen respectivamente 103.92 y 102.61 de índice de polución siendo los primeras 2 ciudades más contaminadas del mundo.

En el ámbito de nuestro país, al 2016 Lima estaba posicionada puesto 19 con 88.59, en el 2017 está en el puesto 25 con 89.32, estos números indican basados en una escala de [0-100] preparada por Numbeo [7], que tan contaminado están los ríos, el aire y la tierra pudiendo ser ocasionado por diversos factores como radiación, metales, gases tóxicos, etc.

La FAO (Food and Agriculture Organization of United Nations) nos ofrece indicadores relacionados a qué tan accesible es la comida, nutrientes y proteínas de la población, además de la superficie agrícola utilizada, las cuales cada año van incrementando junto con la emisión de CO₂ por año [8].

En AQI (Air Quality Index) [9] se puede observar que en diversos lugares de la ciudad de Lima se tiene un índice de polución moderado y en ciertos sectores

como la Embajada de US en Lima esta en el índice "No saludable para cierto grupo de personas ".

Debido a esto, los casos de alta contaminación que afectan la calidad de vida de las personas es casi cotidiano mencionarlo en la ciudadanía Peruana o leerlo en páginas internacionales que realizan estudios de calidad de vida. Ante esto, las autoridades no realizan una concientización sobre nuestra realidad ambiental para poder prevenir el deterioro de nuestro entorno.

Teniendo la situación descrita, en este seminario de tesis se expone como objetivo desarrollar una plataforma web que permite el control y monitorización en tiempo real de diversos tipos de contaminantes u otros parámetros, para así poder asegurar y controlar los lugares o ubicaciones en los cuales existe, se genera o se acumula mayor contaminación.

1.2. Objetivos

El objetivo de este seminario es la de implementar una plataforma web service en la que se pueda realizar el control y monitorización, en tiempo real e histórica, de datos (basados en parámetros) de módulos destinados para el estudio y/o análisis de información obtenidos.

Específicamente, los objetivos de este trabajo con respecto a la plataforma web son:

- Recepción y almacenamiento de datos censados y alertas enviadas/generadas por los Módulos de monitoreo.
- Monitorización en tiempo real de los sensores personalizados y utilizados por cada módulo.
- Monitorización de las posibles alertas o anomalías obtenidas en el censado de información basado en parámetros pre-definidos.
- Registro histórico de los datos censados y alertas.

- Administración y envío de alertas en caso de encontrarse una anomalía fuera de los valores esperados.

Y los objetivos con respecto a las competencias académicas desplegadas en el trabajo son:

- Desarrollo de una plataforma web utilizando Open Software y librerías Open Source.
- Desarrollo Front-end y Back-end de un servicio web para la monitorización en tiempo real de los datos/alertas censadas/generadas.
- Utilización de metodologías de desarrollo de software ágiles y adaptables al cambio continuo.
- Implementación de un RESTful API de información de los datos censados, los cuales serán utilizados por las aplicaciones web y/o móvil para mostrar la data.

1.3. Estructura del Seminario

La estructura del seminario presente se especificará en este punto, para tener una idea clara sobre el contenido del trabajo de seminario de tesis.

■ **Introducción**

En este capítulo se presenta una introducción comentando sobre las motivaciones que han hecho posible el desarrollo de este trabajo, objetivos específicos y generales, se describe la estructura del seminario de tesis.

■ **Estado del Arte**

En este capítulo se presenta un panorama en el que se encuentra envuelto la temática, presentando medidas que se han tomado para solucionar los problemas descritos, así como también, nos brinda una guía para poder discernir cual seria una nueva posible solución.

- **Tecnologías, Herramientas y Metodologías Utilizadas**

En este capítulo se presenta la metodología de desarrollo de software que se utilizó para poder implementar la plataforma web, las tecnologías escogidas y el patrón de desarrollo en el cual está basado.

- **Análisis y Diseño de la Plataforma**

en este capítulo se presenta los actores del sistema, detallándose requisitos que debe cumplir la plataforma web, el modelado de datos y las Reglas de Negocio.

- **Caso de Estudio**

En este capítulo se presenta de manera específica las funcionalidades, el correcto uso de las herramientas implementadas en la plataforma web.

- **Conclusiones y Trabajo a Futuro**

En este capítulo se presenta las conclusiones obtenidas de este trabajo. Adicionalmente, se proponen trabajos a futuro en base al enfoque, implementaciones en la plataforma web.

- **Apéndice A**

En este apéndice se define algunos conceptos que se requieren para entender el presente trabajo.

- **Apéndice B**

En este apéndice se definen las especificaciones de software de cada actor definido previamente en los casos de uso.

- **Apéndice C**

En este apéndice se describe la metodología implementada y también se detallan los Sprint Backlogs generados y utilizados en la metodología de desarrollo de la plataforma.

- **Apéndice D**

En este apéndice se expone una guía de instalación de las herramientas de desarrollo utilizadas.

Capítulo 2

Estado del Arte

En este capítulo se presenta un estudio de el estado actual de la temática que se expondrá en base a los trabajos desarrollados con anterioridad al proyecto con la finalidad de contrastar, mejorar y discernir las ideas, características, funcionalidades, tecnologías y objetivos principales a incluir en este trabajo.

2.1. Introducción

Las tecnologías actuales sobre censado de información en tiempo real son las más necesitadas, debido a la gran cantidad de información que se puede obtener de un lugar o determinada zona con tan solo un par de datos obtenidos del ambiente, así como también, analizar la información obtenida de ese mismo ambiente durante un determinado período de tiempo.

Debido a esta necesidad por conocer las características del ambiente de ciertos lugares, los cuales podrían ser inaccesibles o difícil acceso como campos de cultivo o sino también playas donde se requiera un censado de la intensidad de rayos UV (ultra violeta)[10].

Muy aparte de estos temas ya mencionados, los cuales son censar información en tiempo real y dar un resultado, ya sea en forma de advertencia o sugerencia, de la información analizada segundo a segundo.

En una Ciudad Inteligente se cuenta con una ciudad dividida o segmentada por sectores, los cuales podrían ser separados por tener características similares en temperatura, humedad, etc.

El censado de información en tiempo real conlleva de la mano de manera implícita a buscar un modelado de la información almacenada de tal manera que ocupe la menor cantidad de espacio en disco y mayor velocidad de respuestas al momento de hacer consultas, así como también, un medio de comunicación en el cual no se consuma demasiado ancho de banda.

Hasta la fecha, si bien existe software que realizan el censado en tiempo real, la información generada no es almacenada, ni tampoco analizada posteriormente, por lo cual en muchos casos la información se pierde inhabilitando un posible estudio debido a que no hay data pasada.

2.2. Monitoreo Remoto de un Sistema

En los sistemas de monitoreo remotos se presenta una web que muestra información estática de datos obtenidos, no son optimizadas para ser dinámicas, es decir, en tiempo real.

Las cuales, basados en valores predeterminados, escogen un valor censado mostrándolo según el usuario escoja. Otra manera de realizar el monitoreo es mediante dispositivos móviles, los cuales muestran información de manera similar a la web.

2.2.1. Estado Actual

En el caso del monitoreo remoto [11] se observa que el sistema usando XBee's y arduinos logra la recolección de información de sensores por dispositivo regados en varios sectores de cultivo, pero no muestra los datos recolectados en tiempo real, sino más bien, solo muestra intervalos mínimo y máximo en los cuales los valores se encuentran.

En el caso del monitoreo automatizado [12] se observa que el sistema usando el protocolo zigbee, arduinos y un arduino Mega ADK logra la recolección de información de la humedad de cultivos, pero los datos lo muestra usando Google Drive, el cual muestra la humedad promedio, on/off del estado del sistema y de la alarma.

Pues ahí almacena la información la cual se sobrescribe mostrando el último valor, y en dispositivos móviles, usando el Arduino Mega ADK, el usuario interactúa con el sistema mediante SMS con opciones de prender o apagar el sistema.

En el caso del monitoreo remoto controlado de un sistema de antenas solares [13, 14] se observa que la información obtenida es del voltaje o amperios de las baterías de una casa, las cuales son consultadas vía web y/o móvil usando la tecnología GSM, para el cual tienes que enviarle un mensaje para que te actualice la información, en vez de hacerlo de manera automática.

2.3. Aplicaciones de censado en Tiempo Real

En ya varias partes del mundo se tiene la visión de realizar un censado de información en tiempo real, la cual genere data para poder procesar realizando data mining y realizar mecanismos de prevención utilizando técnicas de machine learning y el campo en el cual están centrando el tema es en la agricultura y temas climáticos.

2.3.1. ClearBlade

Es un software de plataforma web el cual brinda servicios sobre IoT, es decir, puedes crear conectividad con dispositivos cualesquiera, los cuales brindan la información que un usuario quieras y el ClearBlade [1], usando el protocolo MQTT y SDK de desarrollo, te notificará sobre cambios (encendido apagado, data, conectividad) en los dispositivos asociados, es decir, ClearBlade funciona como broker web Socket MQTT.

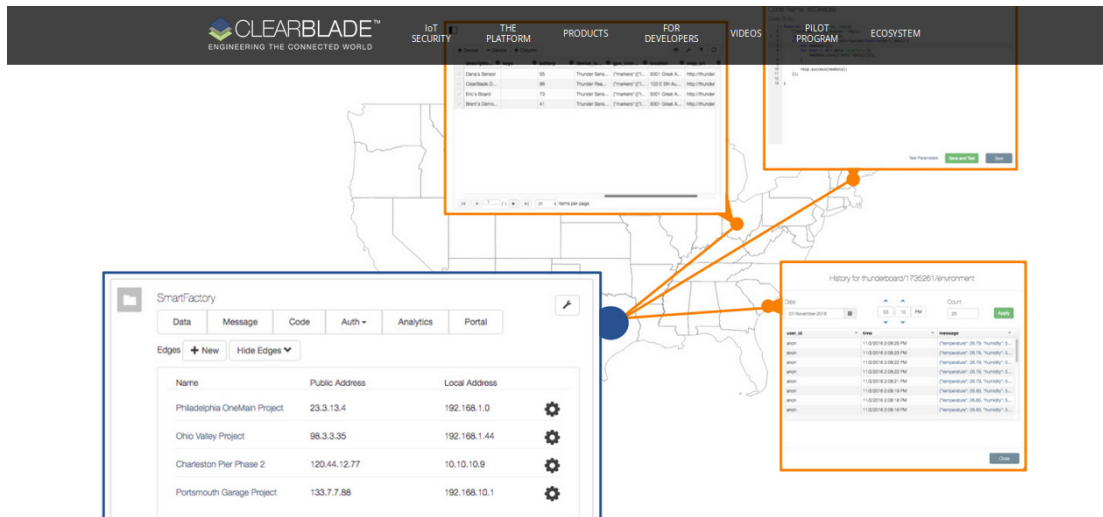


FIGURA 2.1: ClearBlade dashboard, sistema de monitorización de los dispositivos asociados. Fuente: ClearBlade Website. [1]

2.3.2. Hologram

Es un conjunto de dispositivos que se pueden monitorear mediante un dashboard (aplicativo web y móvil), generando una red IoT con módulos que se pueden integrar a arduino y raspberry pi.

Dichos módulos se adaptan utilizando un dispositivo llamado Hologram Dash, el cual tiene el software de comunicación.

Los protocolos que usa Hologram [2] para el envío son bluetooth, Wi-Fi y GSM, los cuales envían información a su plataforma cloud mediante internet, almacenándola para posteriormente se mostrada al dashboard.

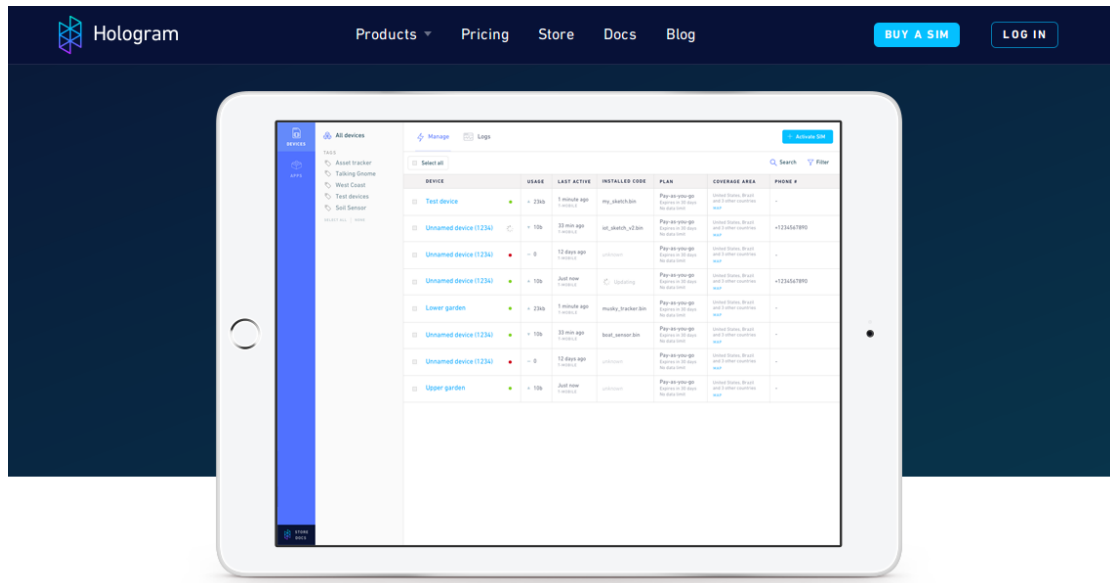


FIGURA 2.2: Hologram dashboard, software destinado al monitoreo en tiempo real. Fuente: Hologram Website. [2]

2.4. Conclusiones

Hemos visto la necesidad de que la información censada en tiempo real, no siempre es almacenada debido a que no tiene un uso en específico para el usuario y una necesidad a nivel global es la información la cual mayormente está sometido a un precio simbólico en algunos casos.

Aparte de esto, la información almacenada no solo resultaría útil para el momento en que se recibe, el cual también es un problema, pues para mostrar la información lo que suelen hacer estos sistemas es realizar una consulta a la base de datos donde se alojan estos datos para posteriormente mostrarlos.

Pues a medida que se va almacenando se puede realizar un análisis (Data Mining) de la información, a su vez aplicar un mecanismo de machine learning para que aprenda el comportamiento de la información (podría tomarlo por fechas).

De esta manera predecir, según las fechas o temporadas, los posibles valores que tendrá cada sensor en cada dispositivo, previniendo de alguna manera (en ser caso negativo) alguna catástrofe o daño.

Por lo tanto, este trabajo se centra en 2 puntos, los cuales son la actualización de información en tiempo real consumiendo pocos recursos, así también, desarrollo de una plataforma en la cual se pueda administrar, visualizar y localiza módulos que censan información en determinados lugares.

Capítulo 3

Tecnologías, Herramientas y Metodologías Utilizadas

En este capítulo se presenta las tecnologías utilizadas, así como también, una descripción de cada una explicando el por qué de su elección para la implementación de esta plataforma.

3.1. Metodologías de Desarrollo

Existen diversas metodologías de desarrollo de software hoy en día, los cuales son prototipado, cascada, incremental, espiral, RAD, entre otros [15]. Con el cual, se ve una tendencia a que los procesos de desarrollo sean cada vez mas óptimos con entregables en menor tiempo.

Por lo cual, los métodos ágiles tienen una ventaja comparativa respecto a los métodos tradicionales de desarrollo [16, 17].

SCRUM [18] es un proceso en el que se aplican periódicamente un mecanismos de control y buenas prácticas para poder formar un entorno de desarrollo y trabajo colaborativo, es decir, en equipos distribuidos o segmentados por tareas específicas [19] generando un avance óptimo en la resolución de tareas.

3.2. Tecnologías y Herramientas de Desarrollo

Es esta sección se expone una descripción concisa de las tecnologías y herramientas las cuales se utilizaron para el desarrollo de la plataforma.

3.2.1. Software Controlador de Versiones

Para llevar a cabo un control e historial de avance en el desarrollo de la plataforma, los cuales incluyen cambios, mejoras, optimizaciones de procesos, etc. Se utilizó la plataforma web Bitbucket que a diferencia de Github, mantiene de manera privada el código fuente de los proyectos.

3.2.2. Sistema de Gestor de Base de datos

Actualmente, uno de los problemas más grandes en el desarrollo del *IoT* y el desarrollo de plataformas *Fog Computing* es la comunicación entre dispositivos o interoperabilidad [20]. Por el cual se busca una manera óptima de organizar la información sin restricciones de almacenamiento.

Dicha manera de almacenamiento es brindado en motores de base de datos no relacionales o *NoSQL*. Así como también, el formato universal entre la comunicación de dispositivos o servicios es el JSON [21] y también el más optimo a comparación con XML [22, 23].

MongoDB

Es un gestor de base de datos *NoSQL* OpenSource orientado a documentos que permite almacenamiento de información en formato JSON. Al ser un modelo *NoSQL* tiene gran ventaja a nivel de consultas, tiempos de ejecución de queries [24]. Además que también tiene alto rendimiento en dispositivos de poca batería y bajo costo [21].

3.2.3. Go language

Es un lenguaje de programación desarrollado por Google con característica principal de tener concurrencia innata [25, 26], motivo por el cual se decidió implementar el core level de la plataforma en dicho lenguaje.

3.2.4. Javascript

Es un lenguaje de programación interpretado diseñado con los paradigmas de procedimientos y orientado a objetos [27] basado en la sintáxis de ECMAScript [28].

Posee diversos frameworks, incluso un RuntimeEnvironment como NodeJS por lo cual Javascript puede ser utilizado para desarrollo front-end y back-end comunmente llamados lado cliente y lado servidor, por lo que Javascript es denominado como lenguaje de desarrollo End-to-End.

En nuestro caso, lo utilizamos para el lado cliente, pues con este diseñados las vistas y realizamos la interacción con el usuario. Javascript se ejecuta en el web browser de manera nativa.

3.2.5. HTML5, CSS3 y Bootstrap

Son, en conjunto, las herramientas necesarias para el diseño front-end de un aplicativo web.

HTML5

Es un lenguaje de marcado que define un conjunto etiquetas las cuales estructuran una página web. HTML es un tipo de documento mediante el cual se puede definir la estructura de una pagina web.

Así como también, dependiendo del framework de desarrollo web, puede embeber sintáxis de código para aplicar funcionalidades tales como bucles repetitivos, consultas, animaciones, etc.

CSS3

Es un lenguaje de diseño gráfico que define y crea una presentación de un documento estructurado (como HTML).

CSS es un lenguaje mediante el cual crea un estilo o animación a una pagina web, con posibilidad de poder ser embebido en documentos HTML y código javascript, así como también en lenguajes de programación como java, python, etc.

Bootstrap

Es un framework de código abierto para el diseño de aplicaciones web. En su haber contiene plantillas de diseño con tipografía, formularios, botones, cuadros, menús de navegación y otros elementos de diseño basado en HTML y CSS, así como también, tiene librerías de JavaScript adicionales.

3.2.6. Google Maps API

Google Maps API es una librería de Google implementada para varias plataformas como Android, iOS y Web. En este caso, se utiliza la librería para web, la cual está implementada en Javascript (debido a que este lenguaje se ejecuta sobre los web browsers).

Con esta librería se permite la manipulación de mapas de Google, ya sea posicionamiento, heatmaps, rutas, polígonos, etc. Esta librería se utiliza para la ubicación de módulos, trazado de sectores que delimiten el alcance, etc.

3.2.7. AngularJS

AngularJS [29] es un framework de desarrollo MVC para el lenguaje Javascript para el desarrollo Front-end de aplicaciones web con el concepto de SPA. AngularJS permite modificar los ficheros HTML con directivas, creando SPA dinámicas.

3.2.8. Protocolos de comunicación

Los protocolos de comunicación son un sistema de reglas que permite que 2 o más entidades o actores puedan entablar un medio de comunicación mediante el cual logren intercambiar información.

Tienen diferentes arquitecturas como Solicitud - Respuesta (en adelante será req-res) o Publica-Suscribe (en adelante será pub-sub) que corresponden a abstracciones como Cliente-Servidor (en adelante Client-Server) o Cliente-Broker (en adelante será Client-Broker).

Simple Mail Transfer Protocol

SMTP [30] se utiliza para el intercambio de mensajes de correo electrónico entre computadoras u otros dispositivos a través de una cola de mensajería.

3.2.9. Protocolos *IoT*

Los protocolos para *Internet of Things* más conocidos son: Coap, MQTT, AMQP e incluido HTTP [31].

Dichos protocolos funcionan en diversos aspectos o necesidades que arquitecturas como *IoT* o *Fog* requieran, ya sea cliente-servidor con un servicio REST como el caso de HTTP y CoAP [32] o en M2M con un servicio pub/sub como MQTT, AQMP y CoAP.

HyperText Transfer Protocol

HTTP [33] permite las transferencias de información en la World Wide Web basado en request/response. También define la semántica y la sintaxis que se utilizan en una arquitectura web entre clientes, servidores y proxies.

Con un payload y cabecera de tamaños indefinidos.

Message Queue Telemetry Transport

MQTT [34] permite la comunicación M2M basado en publica/suscribe [32]. Con un tamaño de cabecera de 2 Bytes y un payload (mensaje) de tamaño máximo de 256 Mb [35].

Constrained Application Protocol

CoAP [36] permite la comunicación M2M basado en request/response y resource/observe (una variación de publica/suscribe) [32] que se ejecuta sobre UDP. Con un tamaño de cabecera de 4 Bytes y un payload de un máximo que depende del servidor web [35].

Advanced Message Queuing Protocol

AMQP [37] permite la comunicación M2M basado en request/response y publica/suscribe. Con un tamaño de cabecera de 8 Bytes y un payload de tamaño indenifido (depende del servidor y sus recursos [6].

Características	MQTT	CoAP	AMQP	HTTP
-----------------	------	------	------	------

1. Creación	1999	2010	2003	1997
2. Arquitectura	Client-Broker	Client-Broker Client-Server	Client-Broker Client-Server	Client-Server
3. Abstracción	Pub-Sub	Req-Res Pub-Sub	Req-Res Pub-Sub	Req-Res
4. Tamaño de cabecera	2 Bytes	4 Bytes	8 Bytes	Indefinido
5. Tamaño de mensaje mín	pequeño	pequeño	pequeño	pequeño
6. Tamaño de mensaje máx	256 MB	Indefinido	Indefinido	Indefinido
7. Métodos	Connect, Disconnect, Publish, Subscribe, Unsubscribe, Close	Get, Post, Put, Delete	Consume, Deliver, Publish, Get, Select, Ack, Delete, Nack, Recover, Reject, Open, Close	Get, Post, Head, Put, Patch, Options, Connect, Delete
8. QoS / Confiabilidad	QoS-0: At most once (Publica y olvida) QoS-1: At least once QoS-2: Exactly once	Confirmable Message: Similar to At most once Non confirmable Message: Similar a At least once	Settle Format: Similar to At most once Unsettle Format: Similar a At least once	Limitado (vía TCP)

9. Puertos por defecto	1883 (TCP) 8883 (TLS)	5683 (UDP) 5684 (DLTS)	5671 (TLS) 5672	80 (HTTP) 443 (HTTPS)
------------------------	--------------------------	---------------------------	--------------------	--------------------------

CUADRO 3.1: Tabla comparativa de características de protocolos de mensajería para sistemas IoT: MQTT, CoAP, AMQP y HTTP.

Fuente: [6]

Según los resultados de [35] y [38] se obtiene que MQTT tiene mejor performance en baja latencia y de confiabilidad de transferencia de datos que CoAP en dispositivos de bajos recursos, así como también XMPP y CoAP tienen escalabilidad horizontal en servidores a diferencia de MQTT que escala verticalmente en dispositivos de bajos recursos.

Además de que la mejor arquitectura de comunicación utilizada para manejar clientes concurrentes en tiempo real orientado a soluciones *IoT* o *Fog Computing* es el Pub/Sub, así como también, en redes de sensores inalámbricas [39].

3.2.10. Patrones de Arquitectura

Un patrón de arquitectura [40] es aquel que brinda una descripción de los elementos y el tipo de relación que tienen, a su vez, un conjunto de restricciones que indican su correcto uso.

MVC

El primer patrón planteado para desarrollar la plataforma web es MVC (Model-View-Controller), que es un patrón de desarrollo para implementar interfaces de usuario en equipos o dispositivos.

Divide una aplicación de software en tres partes interconectadas, con fin de separar las representaciones internas de información que se presenta o se acepta del usuario.

Generalmente se utiliza para el diseño de interfaces graficas de usuario (GUI) en desktop, aplicaciones web y moviles.

Al igual que con otras arquitecturas de software, MVC expresa el "núcleo de la solución "[41].

Componentes

El componente central de MVC, el modelo, captura el comportamiento de la aplicación en términos del dominio del problema, independientemente de la interfaz de usuario.

- **Model:** Esta capa maneja directamente la data, lógica y las reglas de negocio de la aplicación.
- **View:** Esta capa es cualquier representación de información. Se puede mostrar múltiples vistas de una misma información basadas en cambios en el modelo.

- **Controller:** Esta capa envía consultas al modelo y lo actualiza. También envía consultas a la vista asociada cambiando su presentación.

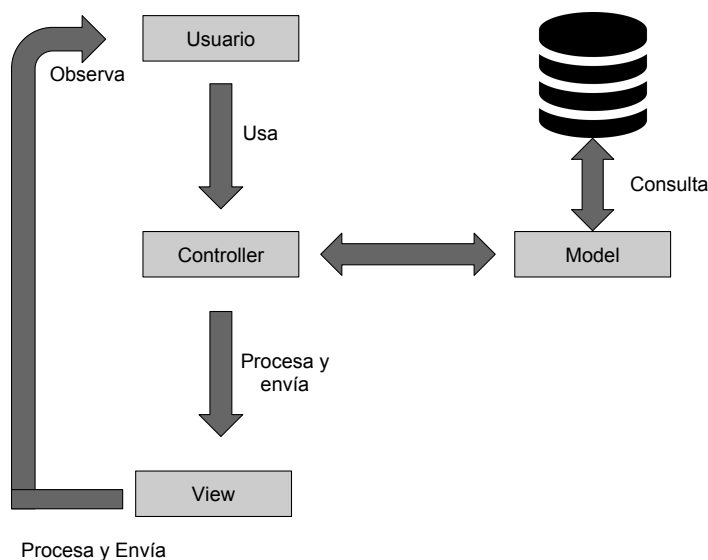


FIGURA 3.1: Patrón Model-View-Controller. Fuente: Wikipedia MVC. [3]

MVW

El segundo patrón planteado para desarrollar la plataforma web es MVW (Model-View-Whatever), que es un patrón de desarrollo para implementar interfaces de usuario en diversos dispositivos.

Hace una división de la aplicación de Software en más de 3 capas, donde a la tercera capa, Whatever, se le puede implementar o inyectar más niveles de abstracción.

Esta arquitectura de Software se basa en el dinamismo de AngularJS, pues está compuesto por directives, factories, filters, routers, etc [29]

Componentes

El componente central de MVW, el modelo interactúa con la o las capas de Whatever, las cuales posteriormente son presentadas en el View.

- **Model:** Esta capa maneja directamente la data, lógica y las reglas de negocio de la aplicación.
- **View:** Esta capa es cualquier representación de información. Se puede mostrar múltiples vistas de una misma información basadas en cambios en el modelo.
- **Whatever:** Esta capa se puede inyectar o incluir otros tipos diversos de conceptos, como controllers, view-models, presenters, routers, etc.

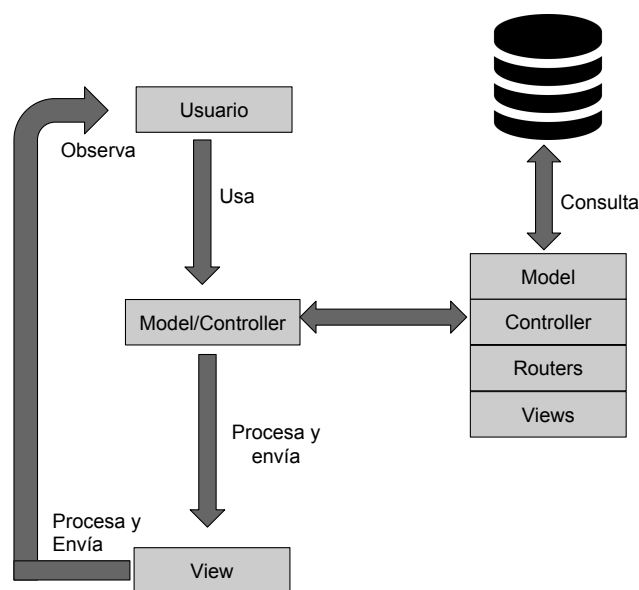


FIGURA 3.2: Patrón Model-View-Whatever. Fuente: AngularJS, a MVW Framework. [4]

Capítulo 4

Análisis y Diseño de la Plataforma

En este capítulo se presenta un detalle específico de la plataforma con sus respectivos actores que forman parte, requisitos, las funcionalidades, el diseño de la arquitectura de servicios y el modelamiento de la base de datos.

4.1. Análisis de la Plataforma

En esta sección se expone los actores, requisitos de la plataforma y reglas de negocio.

4.1.1. Actores

Son las entidades, no necesariamente humanas, que interactúan con la plataforma sin ser parte de la mismo.

En esta plataforma se identifica 3 roles o actores bien marcados. Un actor puede ser una o más entidades que realizan unas acciones similares, es decir, realiza el mismo rol.

Los roles de los actores son:

- **Administrador:**

Este actor es el encargado de la administración de los usuarios y también la monitorización de los demás actores que conforman todo el entorno de la plataforma. Es el único usuario que puede modificar datos o información, además de poder monitorear todos los módulos de todos los usuarios.

■ **Usuario:**

Es el actor que solicita servicio a la plataforma, es decir, su nivel de acceso esta limitado a sólo poder ver y monitorear módulos específicos que estén asociados a el.

■ **Módulo:**

Es el actor encargado de censar y enviar información a la plataforma, está asignado a una determinada área a cubrir.

4.1.2. Reglas de Negocio

En las reglas de negocio definen los parámetros que se debe tener en cuenta en la implementación de la plataforma, las cuales son:

- Solo existe un único usuario con privilegios de administrador en la plataforma, el cual puede manipular, modificar y/o eliminar información.
- Los módulos instalados enviarán la información censada de cada uno de los sensores que tengan acoplados mediante los protocolos especificados. Estos serán en formato JSON.
- La información de los módulos a registrar son el modelo, posición (latitud y longitud), área a cubrir, definir una zona, asociar a un sector y un listado de sensores los cuales tendrán información como tipo de sensor, modelo del sensor, unidad de medición y su estado.
- Los usuarios solicitarán el servicio para que el administrador le asigne (con previa aprobación) un usuario y contraseña, además el usuario deberá especificar en qué ubicación y en qué sensores estaría interesado. Para que posteriormente pueda acceder al dashboard y ver los datos en tiempo real de los módulos escogidos.
- La información personal requerida de los usuarios son: DNI, Nombres, Apellidos, Email, Usuario, Contraseña, módulos asociados y una fotografía.

- La plataforma monitorizará los datos censados en tiempo real y mostrará en el dashboard según vaya captando datos.
- La plataforma mostrará información detallada a cada usuario según sus dispositivos exclusivamente asociados y asignados, El único que puede monitorear todos los módulos de todos los usuarios es el administrador, que es el único que tiene acceso a toda la información generada por todos los módulos.

4.1.3. Requisitos de la Plataforma

En la sección anterior, se explicó sobre la metodología de desarrollo SCRUM, donde se define los Sprint Backlogs (basado en las historias de usuario).

En este caso, el actor que hará uso de todas funcionalidades de la plataforma es el administrador, los demás usuarios están limitados a visualizar.

Por lo tanto se presenta los requisitos finales que han ido cambiando a través del desarrollo, optimizándose, cambiando algunas cosas en cada reunión de entrega de sprint.

Historias de Usuario Finales

En las historias de usuario se describe los requisitos que debe de cumplir la plataforma (en SCRUM son utilizados en los Sprint Backlogs).

- **Administración de Módulos**
 - Se necesita registrar Módulos, especificando a qué sector está asociado, la zona, su ubicación y el área que abarca su censado.
 - Al momento de registrar módulos, se debe poder especificar o modificar los sensores que contiene, con sus respectivos datos siempre especificando el tipo de sensor.

■ **Administración de Usuarios**

- Se necesita registrar Usuarios (sin privilegios de administrador) almacenando sus datos personales.
- Al momento de registrar usuarios, se debe poder asignar o asociar los módulos de los cuales podrá monitorear información censada por los sensores en tiempo real.

■ **Monitorización de los datos**

- El administrador debe ser el único que puede visualizar la data censada de todos los módulos con todos los sensores en tiempo real. Los usuarios solamente podrán monitorizar a los módulos asociados a cada uno.
- El administrador y los usuarios deben poder de ver las ubicaciones de los módulos asociados (el administrador podrá ver todos sin excepción) en un mapa, el cual mostrará también el área o zona que cubre cada módulo de censado.
- El administrador y usuarios deben poder visualizar un histórico de datos censados y almacenados, deben poder escoger una fecha y hora específicos además de un número de datos a observar entre esas fechas/horas escogidas.
- El administrador y los usuarios deben poder ver un mapa de calor en base a los datos censados en tiempo real, el cual mediante una escala gráfica determine si está en una zona segura o no segura.

En el Apéndice B se expone los casos de uso y las especificaciones de software adaptadas a estas historias de usuario.

En el Apéndice C se exponen todos los Sprint Backlogs utilizados al momento de desarrollar, mediante entregas parciales, la plataforma.

4.1.4. Modelo de Datos

Como se expuso en el capítulo 3, sección de Tecnologías y Herramientas de desarrollo, la plataforma utilizará como DBMS a MongoDB.

MongoDB es una Base de Datos No Relacional y posee unas características que lo hace muy útiles para el diseño de base de datos que son difíciles de modelar en modelos relacionales.

Una de las características es que la información lo almacena en documentos, los cuales tienen el formato JSON, que son definidos por objetivos denominados BSON.

Otra característica que pose MongoDB es el alto nivel de concurrencia al momento de insertar, modificar y/o eliminar documentos o colecciones (conjunto de documentos), muy aparte de que no tiene una definición estricta del formato de un documento, es decir, si un documento es insertado con X atributos, un usuario podría insertar un documento con Y atributos en la misma colección (siendo X diferente de Y).

Por lo cual MongoDB al permitir una flexibilidad al momento de almacenar datos, su velocidad de consulta y su concurrencia al accionar sobre los documentos y/o colecciones, se debe inferir que MongoDB es una base de datos que está enfocada a tratar con mejor optimización los cuellos de botella ocasionados por múltiples consultas a la vez.

Debido a esto, MongoDB es la base de datos no Relacional que se escogió basándose en la enorme cantidad de datos que llegarán en simultáneo de los diversos módulos con diversos sensores.

4.2. Diseño de la Plataforma

En esta sección se presenta el diseño y modelado de datos, así como también, la arquitectura de servicios utilizados para la plataforma (tanto como la comunicación y diferencia entre capas de servicio).

4.2.1. Estructura de la base de datos

Esta plataforma está diseñada principalmente para los usuarios, por lo que toda la información debe circular entorno a los usuarios. Teniendo en cuenta las reglas de negocio y los requisitos, se puede realizar un diagrama de Entidad-Relación del modelo de datos.

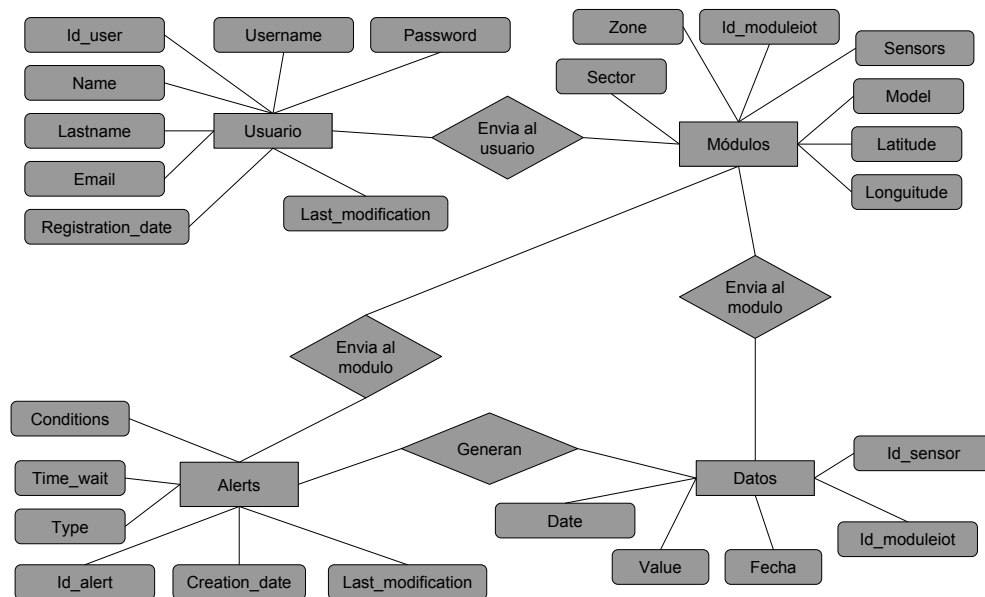


FIGURA 4.1: Diagrama Entidad-Relación. Fuente: Elaboración propia.

El diseño se ha realizado para que la plataforma cumpla con el objetivo de almacenar y tener control de los usuarios, los módulos, la información de cada uno, quienes se asocian entre sí, las relaciones usuario-módulos.

Este diagrama de entidad-relación nos sirve para ver de manera general la estructura de la plataforma, pero lo que se utilizará para la creación de la base de datos es el diagrama de clases, que será la guía para la implementación, tal como se muestra en la figura 4.2.

Como hemos explicado, se implementará una base de datos no relacional orientado a documentos, es decir, no existen las tablas, no hay operaciones por join, no hay restricción en el dinamismo de insertar datos.

Debido a esto, MongoDB puede redundar datos como quiera, pues MongoDB almacena datos según el ObjectId (que almacena la fecha y hora en que se inserta un documento) que ignora si los documentos insertados son iguales o no (no lo son, se diferencian en el ObjectId) permitiendo una gestión de datos más flexible.

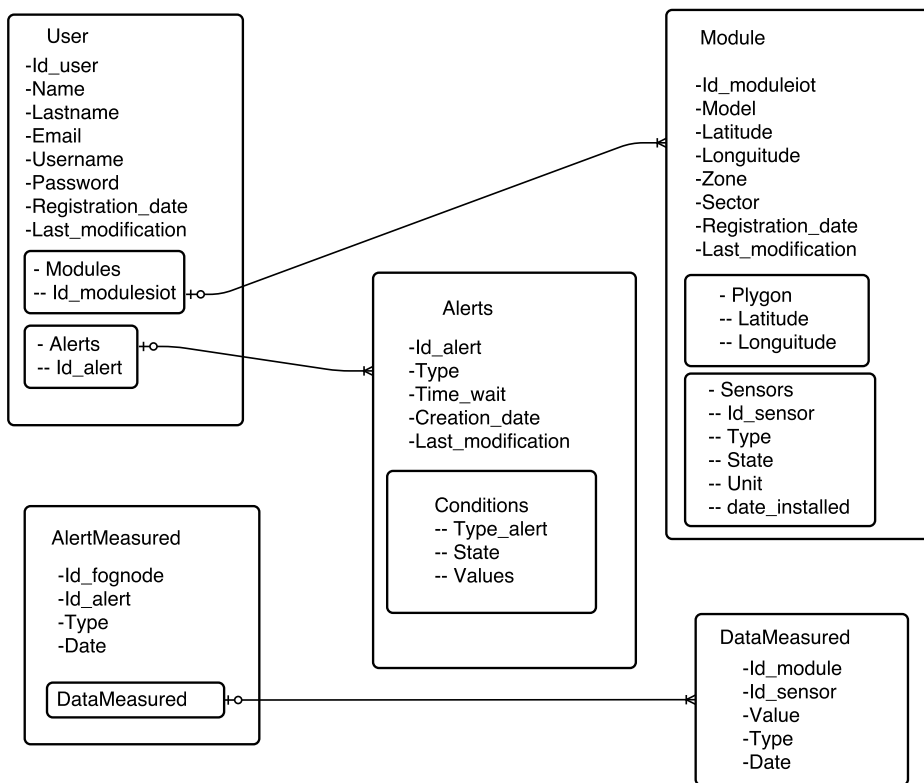


FIGURA 4.2: Diagrama de Clases. Fuente: Elaboración propia.

4.2.2. Diseño de la arquitectura de servicios

Esta plataforma está diseñada utilizando arquitecturas Fog Computing. Tal como he explicado en el Apéndice A, hago una diferencia entre servicios web basados en SOAP y REST. Exxplicaré el por qué de esta arquitectura y no solo la clásica arquitectura de Cloud Computing.

Los servicios web mediante HTTP, se realizan por medio de request-response, es decir, un cliente hace una petición a un determinado

servidor y este recibe la petición, la atiende y envía una respuesta al cliente, la cual depende de qué tipo de petición haya realizado.

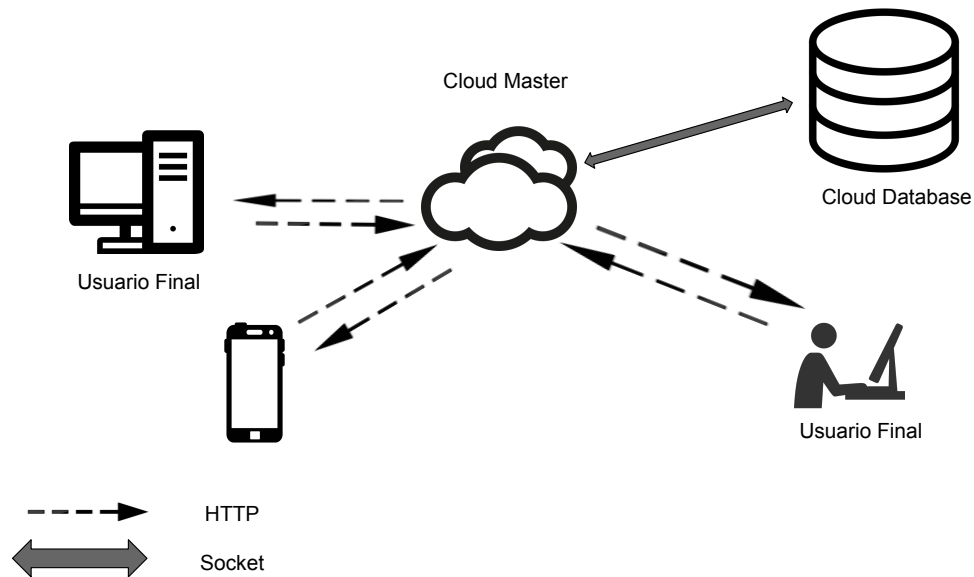


FIGURA 4.3: Representación de servidor Cloud que realiza un servicio web basado en REST. Fuente: Elaboración propia.

Las peticiones para visualizar la data por ejemplo (una de nuestras funcionalidades) se realiza mediante una petición GET usando HTTP, una vez en el servidor cloud gestiona la petición y éste a su vez hace una conexión a la base de datos mediante sockets, si la consulta es válida retornará suceso o en caso contrario, error.

Hasta este punto, nuestro servidor cloud podría tolerar las múltiples consultas que se puedan realizar solicitando información, gracias a la concurrencia innata de MongoDB y el lenguaje Go.

Ahora para el monitoreo en tiempo real de los datos censados, utilizamos el protocolo MQTT, el cual levanta un servicio (llamado broker) que está a la espera de los mensajes enviados y categorizados por tópicos convirtiéndose en una aplicación IoT [42].

Es decir, los dispositivos que envían mensajes a través del broker son denominados publicadores, los que están a la espera de dichos mensajes se denominan suscriptores y a los identificadores (que pueden ser una cadena similar a una URL) son los tópicos.

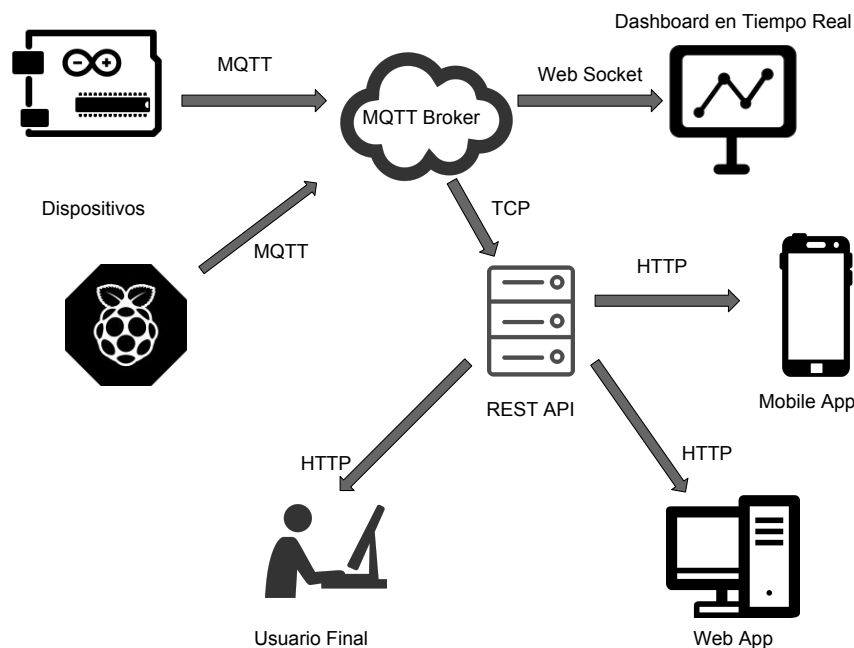


FIGURA 4.4: Representación de un servidor Cloud que funciona como Broker MQTT. Fuente: Elaboración propia.

Sin embargo, a mayor número de dispositivos, el servidor Cloud tendrá que atender mayor número de mensajes por publicar a los diversos suscriptores. Resultando esto desfavorable para nuestra plataforma, pues su tiempo de respuesta será más lento debido a que MQTT se basa en una cola de mensajes y HTTP son petición-respuesta y al combinarlos, generaría un cuello de botella en las respuestas del servidor.

Por lo cual se plantea una arquitectura Fog Computing, la cual consiste en dividir el flujo de información en niveles, creando servidores intermedios, los cuales realizan un poco del procesamiento aligerando la carga y descentralizando las operaciones del servidor Cloud.

En otras palabras, se crearían servidores intermedios, en la arquitectura que se plantea en este trabajo los servidores Fogs serán los que realicen el trabajo de brokers, es decir, el servidor Cloud delega el trabajo a cada servidor Fog ahorrando así poder de cómputo atendiendo los datos recepcionados por el broker MQTT.

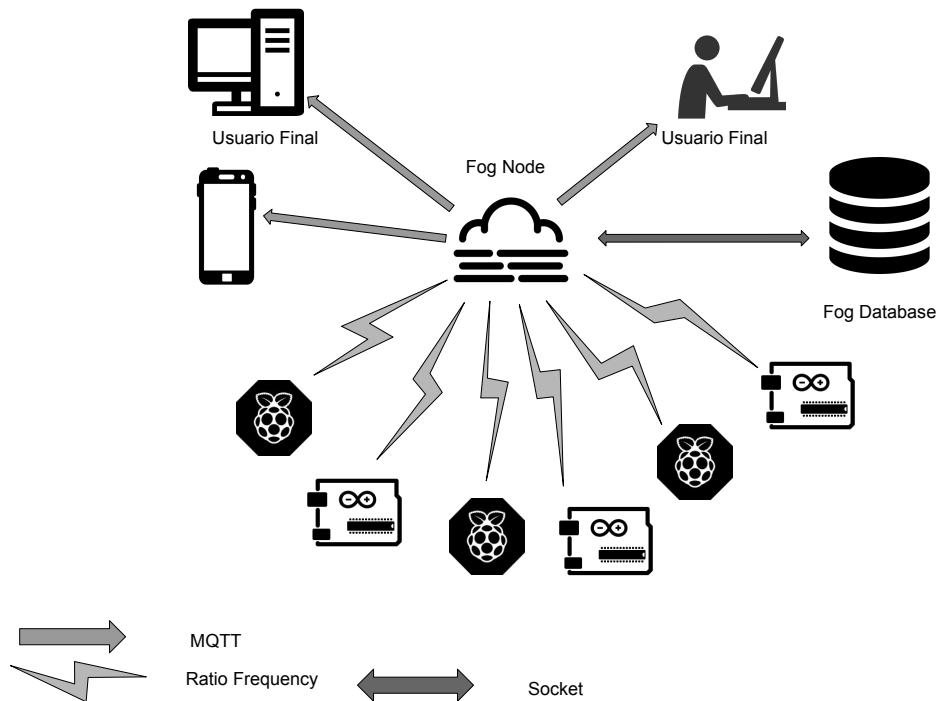


FIGURA 4.5: Representación de un servidor Fog que funciona como Broker MQTT. Fuente: Elaboración propia.

Como se observa en la figura 4.5, los módulos y diversos dispositivos envían los datos censados hacia los servidores Fog a los cuales están asociados y asignados y estos, actuando como brokers MQTT, publican los mensajes hacia los suscriptores que observan y visualizan la data en tiempo real.

Con esta arquitectura [43, 44], se descentraliza los servicios, permitiendo al servidor Cloud resolver peticiones exclusivas del servicio RESTful y a los servidores Fog resolver, publicar y suscribir mediante tópicos a los clientes MQTT que se conectan mediante sockets y/o websockets (cliente móvil y/o web).

Las comunicaciones entre el servidor Cloud y los servidores Fog son mediante 2 protocolos como se puede observar en la siguiente imagen:

■ SSH

Se utilizará el protocolo SSH para realizar conexiones entre servidores y además de enviar las credenciales (generadas utilizando OpenSSL) para realizar conexiones seguras a MQTT sobre el protocolo TLS/SSL evitando así posibles intrusos [45].

■ MQTT

Se utilizará el protocolo MQTT para realizar envíos de información cada vez que se actualice la información correspondiente a un servidor Fog en específico (mediante MQTT se envía la modificación) para el cual, el servidor Fog al recibir el mensaje, modificará su base de datos interna. Actualizándose y sincronizándose con la información del servidor Cloud.

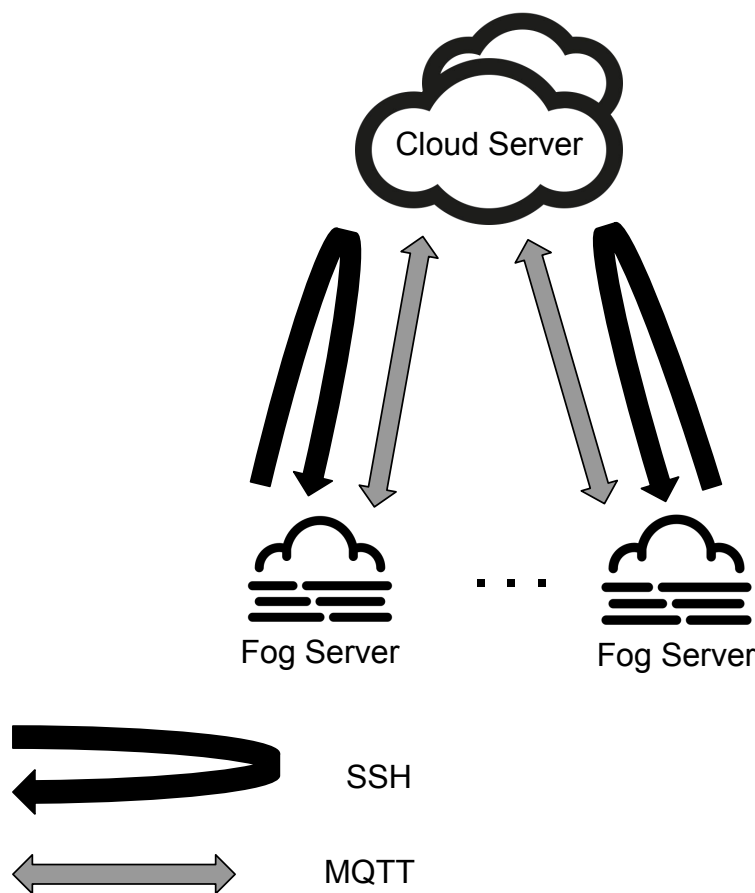


FIGURA 4.6: Representación de un servidor Cloud y diversos servidores Fog, los cuales pueden comunicarse mediante SSH o MQTT. Fuente: Elaboración propia.

Entonces, planteando un panorama que abarque todo, nuestra arquitectura quedaría descrita como un servidor Cloud que recibe información tratada y los servidores Fog que realizan acciones intermedias funcionando como un gateway de un micro datacenter que funciona como un broker MQTT [46], quedando descrito de la siguiente manera:

- El servidor Cloud se encargará del almacenamiento, de la plataforma web y del servicio RESTful de información, tendrá un cliente MQTT con el cual podrá intercambiar información con los servidores Fog siempre y cuando sea necesario.
- Cada servidor Fog se encargará de ser un broker MQTT para todos los módulos y dispositivos asociados a ellos, a su vez publicando en los clientes móviles o web que estén suscritos a tópicos específicos, de tal forma que reciban y visualicen la data en tiempo real.

Por lo que nuestra arquitectura, vista de manera global sería la siguiente:

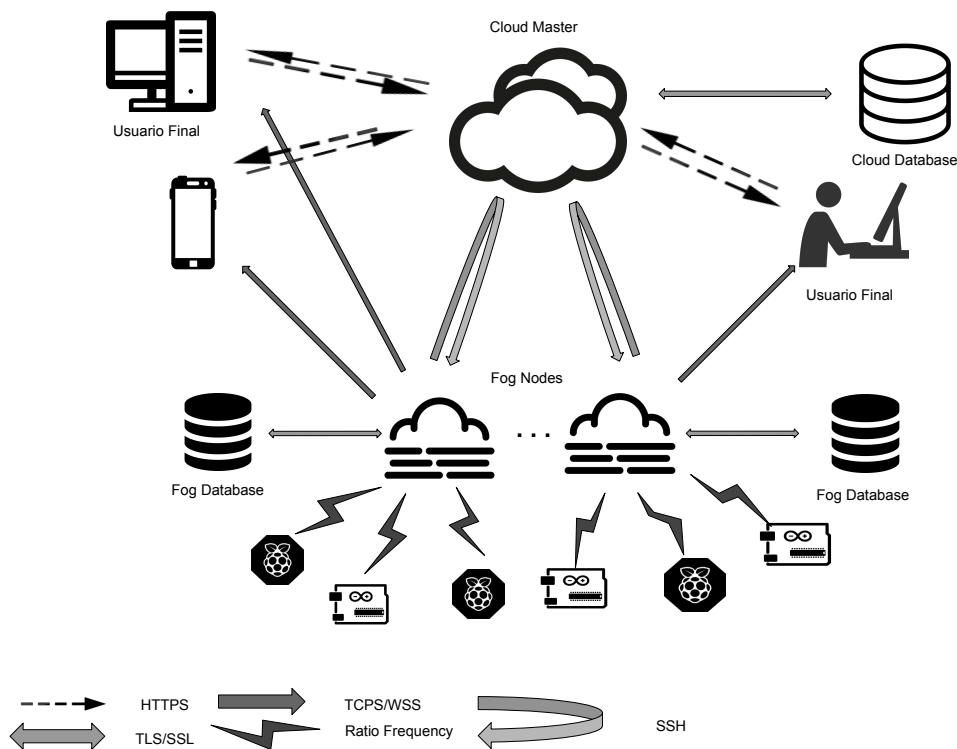


FIGURA 4.7: Representación de la arquitectura Fog Computing de la plataforma. Fuente: Elaboración propia.

Capítulo 5

Caso de Estudio

En este capítulo se presenta las funcionalidades de la plataforma, cómo utilizarla y cómo se deben de interpretar las interacciones con el.

5.1. Inicio de sesión

La primera funcionalidad es la que se presenta al momento de acceder sesión, sólo los usuarios registrados pueden iniciar sesión. Actualmente, se cuenta con 3 usuarios, donde solo uno tiene privilegios de administrador.

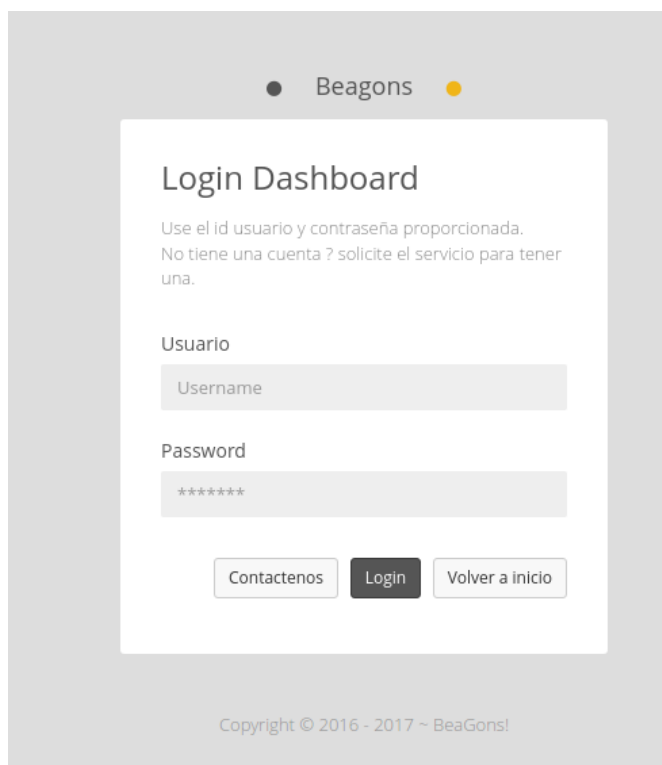
The image shows a login interface for a platform called 'Beagons'. At the top, there's a header with two colored circles (grey and yellow) and the text 'Beagons'. Below this is a white box with the title 'Login Dashboard'. Inside the box, there's a message: 'Use el id usuario y contraseña proporcionada. No tiene una cuenta ? solicite el servicio para tener una.' Below the message are two input fields: 'Usuario' with a placeholder 'Username' and 'Password' with a placeholder '*****'. At the bottom of the box are three buttons: 'Contactenos', 'Login' (which is dark grey), and 'Volver a inicio'. Below the white box, at the bottom of the grey background, is the copyright text: 'Copyright © 2016 - 2017 ~ BeaGons!'.

FIGURA 5.1: Inicio de Sesión. Fuente: Elaboración propia.

Una vez iniciado sesión, se muestra la vista home, donde se muestra un sideNavBar al lado derecho, el cual muestra las opciones el administrador.

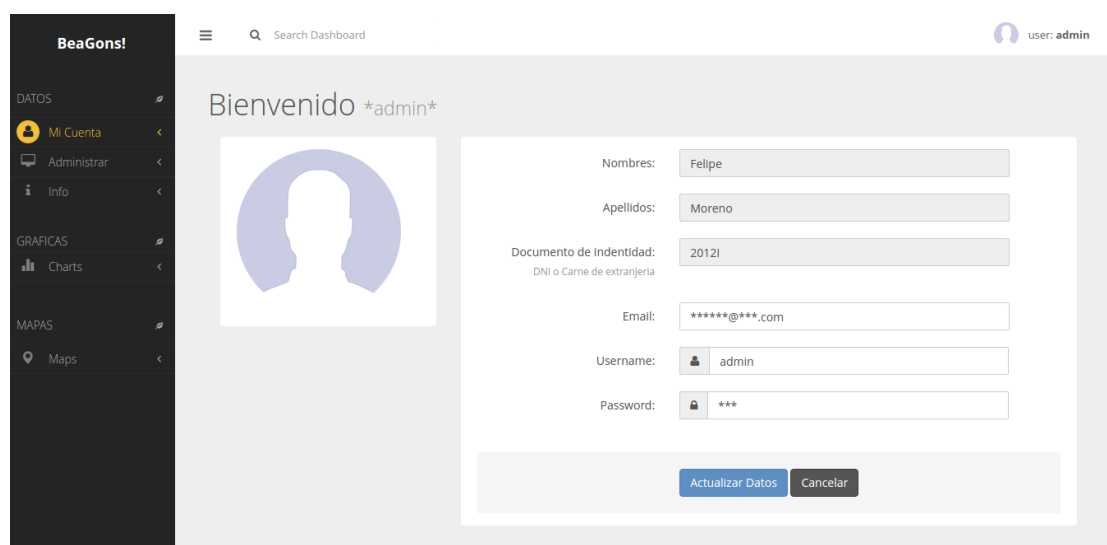


FIGURA 5.2: Home del administrador. Fuente: Elaboración propia.

Y también las opciones que muestran a un usuario cualquiera.

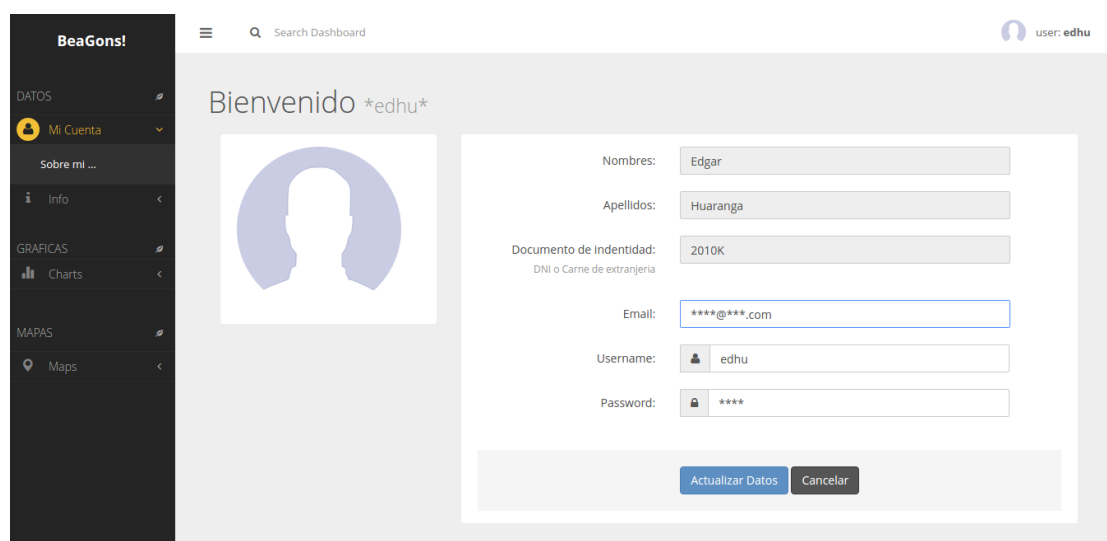


FIGURA 5.3: Home de un usuario. Fuente: Elaboración propia.

Como se ve, la opción de **Administrar** solo es concedido al administrador.

En panel del lado derecho se puede observar la lista de funcionalidades a las que tiene acceso el usuario/administrador:

- **Mi Cuenta:** Muestra la información del usuario.
- **Administrar:** Muestra las opciones para crear, modificar y eliminar usuarios y módulos. (Solo disponible para el administrador).
- **Informacion:** Muestra la información de los usuarios (solo disponible para el administrador) módulos y sensores.
- **Graficas:** Muestra los datos censados en tiempo real de cada módulo, de cada sensor por módulo y un histórico de datos.
- **Mapas:** Muestra las localizaciones de los módulos, así como también, los mapas de calor en tiempo real de los datos obtenidos por los sensores de cada módulos.

5.2. Mi Cuenta

La primera funcionalidad que tiene un usuario o el administrador es la de poder modificar sus datos personales mostrados en la figura 5.3.

5.3. Administración de usuarios

La segunda funcionalidad es exclusiva del administrador es la creación/modificación y eliminación de usuarios.

Al entrar a la opción de Administrar ->Usuario ->Añadir/Modificar/Eliminar usuarios, para el caso de añadir, rellenar los datos solicitados del usuario y los módulos a los cuales estará relacionado.

Así como también puede ver la opción de modificar y eliminar. Para ambos casos, deberá escoger qué usuario (por su Id) desea eliminar/modificar, bastará con apretar el botón eliminar/modificar para realizar dichos cambios.

The screenshot shows the 'Registrar usuario' form in the BeaGons! system. The left sidebar contains a menu with 'Administrar' expanded, showing 'Usuario' with sub-options 'Añadir usuario', 'Modificar usuario', 'Eliminar usuario', and 'ModuleIoT'. The main content area has a breadcrumb 'Se encuentra en: > Registrar usuario' and a title 'Registrar usuario'. The form fields are: 'Nombres' (nombres), 'Apellidos' (apellidos), 'Documento de identidad: DNI o Carne de extranjería' (documento de identidad), 'Email' (email), 'Username' (Username), 'Password' (Password), 'Fecha' (2017-06-20), 'Hora' (16:38:49), and 'Modules: Asociados' (radio buttons for C001, C002, C003, C004). At the bottom are 'Registrar Usuario' and 'Cancelar' buttons.

FIGURA 5.4: Crear Usuarios. Fuente: Elaboración propia.

The screenshot shows the 'Eliminar usuario' form in the BeaGons! system. The left sidebar is the same as in Figure 5.4. The main content area has a breadcrumb 'Se encuentra en: > Eliminar usuario' and a title 'Eliminar usuario'. The form includes a dropdown for 'Id Usuario' (U0002) and fields for: 'Nombres' (Giovanny), 'Apellidos' (Mondragon), 'Documento de identidad: DNI o Carne de extranjería' (2011B), 'Email' (giovanny1335@gmail.com), 'Username' (vgio), 'Password' (123456), 'Fecha' (2016-05-09), 'Hora' (21:44:10), and 'Modules: Asociados' (checkboxes for C001, C002, C003, C004). At the bottom are 'Eliminar Usuario' and 'Cancelar' buttons.

FIGURA 5.5: Eliminar Usuarios. Fuente: Elaboración propia.

5.4. Administración de Módulos

La tercera funcionalidad es exclusiva del administrador es la creación/modificación y eliminación de módulos.

Al entrar a la opción de Administrar ->Usuario ->Añadir/Modificar/Eliminar modulesIoT, para el caso de añadir, rellenar los datos solicitados del módulo, como su modelo, su posición y que área va a delimitar, así como también, los sensores asignados a el.

Así como también puede ver la opción de modificar y eliminar. Para ambos casos, deberá escoger qué módulo (por su Id) desea eliminar/modificar, bastará con apretar el botón eliminar/modificar para realizar dichos cambios.

The screenshot displays the 'Registrar' form in the BeaGons! application. The sidebar on the left shows the 'Administrar' menu expanded, with 'Añadir ModuleIoT' selected. The form fields include:

- Modelo:
- Latitud:
- Longitud:
- Zona:
- Sector:
- Fecha:
- Hora:
- Pos:
- Pos:
- Añadir pos:
- Quitar pos:
- Sensores:

Tipo sensor
modelo de sensor
state
unidad
18:14:06
2017-06-20
- Añadir sensor:
- Quitar sensor:

At the bottom of the form are two buttons: 'Registrar IoTDevice' and 'Cancelar'.

FIGURA 5.6: Crear Módulos. Fuente: Elaboración propia.

Se encuentra en: > Eliminar IoTDevice

Eliminar

Id ModuleIoT: C004

Modelo: verl

longitud: -77.0505176

latitud: -12.0161385

Zona: Puerta 6

Sector: FIIS

Fecha: 2016-09-20

Hora: 13:44:10

Sensores:

- Dioxido
- CO2
- free
- ppm
- 2016-05-09 21:44:10

Eliminar IoTDevice Cancelar

FIGURA 5.7: Eliminar Módulos. Fuente: Elaboración propia.

5.5. Información

La cuarta funcionalidad que tiene un usuario o administrador es la información de los usuarios (sólo para el administrador), los módulos a los cuales el usuario está asociado y los sensores de los módulos de cada usuario.

Beagons!

Search Dashboard

user: admin

lista de **Usuarios**

Se encuentra en: > Info Usuarios

Datos de Usuarios

ID USUARIO	USERNAME	NOMBRE	APELLIDO	DOC IDENTIDAD	EMAIL	FECHA DE REGISTRO
U0000	admin	Felipe	Moreno	2012I	dohko_libra1995@hotmail.com	2016-05-09 21:44:10
U0001	edhu	Edgar	Huaranga	2010K	edhu1227@gmail.com	2016-05-09 21:44:10
U0002	vglo	Giovanny	Mondragon	2011B	giovanny1335@gmail.com	2016-05-09 21:44:10

FIGURA 5.8: Información de los Usuarios. Fuente: Elaboración propia.

BeaGons! user: admin

Search Dashboard

lista de ModuleIoT

Se encuentra en: > Info ModuleIoT

Id de Usuario:

Datos de ModuleIoT

ID MODULEIOT	MODELO	ZONA	LONGITUD	LATITUD	FECHA DE INSTALACION
C001	verl	Puerta 5	-77.0507113	-12.0173721	2016-05-09 21:44:10
C003	verl	Puerta 5	-77.0493128	-12.0172765	2016-09-20 13:44:10
C004	verl	Puerta 6	-77.0505176	-12.0161385	2016-09-20 13:44:10

FIGURA 5.9: Información de los Módulos. Fuente: Elaboración propia.

BeaGons! user: admin

Search Dashboard

lista de Sensores

Se encuentra en: > Info Sensores

Datos de Sensores

ID MODULEIOT	ID SENSOR	TIPO SENSOR	UNIDAD	MODELO	FECHA DE INSTALACION
C001	S50001	Temperatura	°C	T1	2016-05-09 21:44:10
C001	S51001	Presion	Pa	PA1	2016-05-09 21:44:10
C001	S52001	Monoxido	ppm	CO1	2016-05-09 21:44:10
C002	S50002	Temperatura	°C	T1	2016-05-09 21:44:10
C002	S51002	Presion	Pa	PA1	2016-05-09 21:44:10
C002	S53001	Dioxido	ppm	CO2	2016-05-09 21:44:10
C003	S50003	Temperatura	°C	T1	2016-05-09 21:44:10
C003	S53002	Dioxido	ppm	CO2	2016-05-09 21:44:10
C004	S53003	Dioxido	ppm	CO2	2016-05-09 21:44:10

FIGURA 5.10: Información de los Sensores. Fuente: Elaboración propia.

5.6. Graficas

La quinta funcionalidad que tiene un usuario o administrador es la visualización de data, la cual puede ser de 3 maneras:

■ Modules

Aquí se muestra la data conforme escojas el módulo, muestra la información censada de todos los sensores por cada módulo.

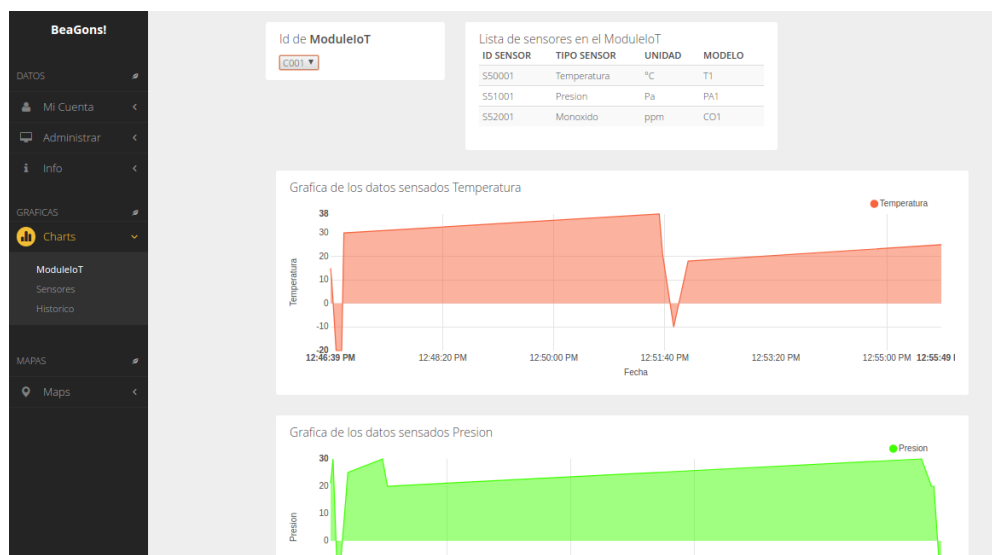


FIGURA 5.11: Visualización seleccionando por Modulo. Fuente: Elaboración propia.

■ Sensores

Aquí se muestra la data conforme escojas el tipo de sensor, puedes escoger un módulo específico en base a eso o , muestra la información censada de todos los sensores por cada módulo.

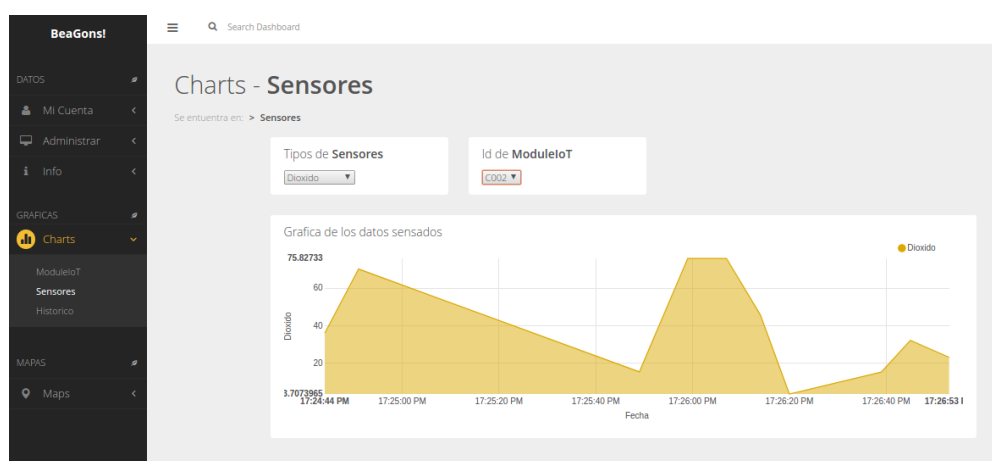


FIGURA 5.12: Visualización seleccionando por tipo sensor. Fuente: Elaboración propia.

■ Historico

Aquí se muestra la data de manera histórica, es decir, se puede revisar en base a fechas y horas, así como también, los últimos X datos, donde X es una opción a escoger.



FIGURA 5.13: Historico de datos. Fuente: Elaboración propia.

5.7. Mapas

La sexta funcionalidad que tiene un usuario o administrador es la visualización de las ubicaciones de los módulos, es decir, pueden saber dónde se encuentran con exactitud los dispositivos asociados al usuario.

Se muestra 2 tipos de mapas, uno exclusivamente de ubicaciones y otro que muestra los mapas de calor generados por la data obtenida en tiempo real.

■ Modules Locations

Muestra las posiciones de los módulos junto con las áreas que cubren respectivamente, es decir, que la data censada es por toda el área cubierta.

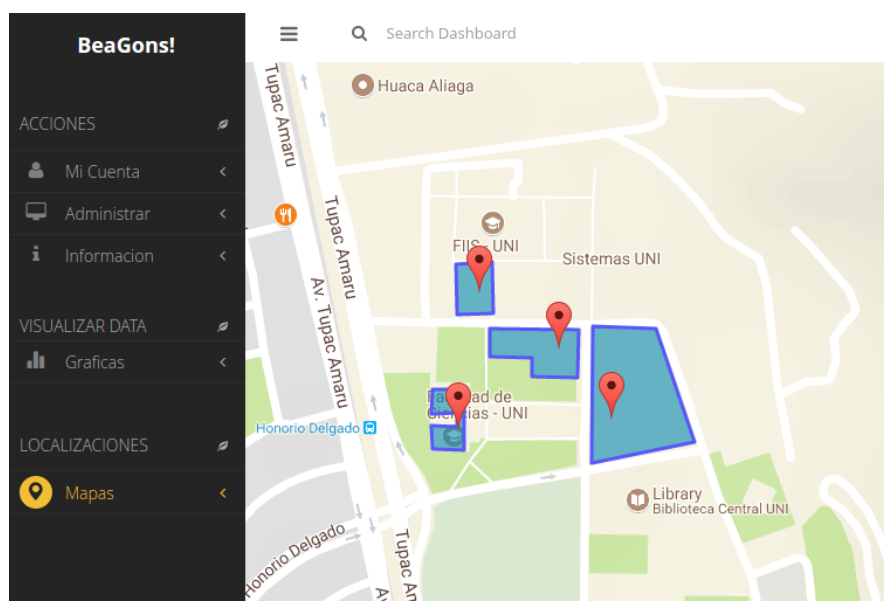


FIGURA 5.14: Ubicacion de los módulos junto con el área que cubren. Fuente: Elaboración propia.

■ HeatMaps

Muestra los módulos y el mapa de calor (que va cambiando en tiempo real según el valor censado) en las áreas cubiertas por los módulos.



FIGURA 5.15: Mapas de calor de los valores censados representados en las áreas de los módulos. Fuente: Elaboración propia.

Capítulo 6

Conclusiones y Trabajo Futuro

En este capítulo se presenta las conclusiones respecto a los objetivos planteados al inicio del seminario, los cuales han sido explicados detalladamente en los capítulos anteriores, presentando algunas inferencias finales para este trabajo.

6.1. Conclusiones

Se ha propuesto esta plataforma de monitoreo en tiempo real para realizar un seguimiento y control sobre el ambiente climático.

La plataforma desarrollada cumple con los requisitos que se propusieron al inicio de este trabajo. Aunque aún queda trabajo que podrían derivar de este.

Por lo tanto, se presenta las siguientes conclusiones:

- **Recepción de datos y alertas:** La plataforma está permanentemente a la espera de los datos que son enviados por los módulos. Para esto, es necesario que los módulos envíen constantemente datos y/o alertas generadas.
- **Alertas:** La plataforma está en un estado constante de evaluación de los parámetros definidos, los cuales generan alertas de anomalías, para anunciar en caso ocurra algún imprevisto.
- **Monitorización en tiempo real:** La plataforma, basada en su arquitectura, brinda información en tiempo real mientras a la vez lo almacena generando histórico de datos para la visualización.

- **Datos Históricos:** La plataforma brinda la funcionalidad de poder ver un histórico de datos que han sido censados a lo largo de un período a especificar dando la posibilidad de observar cambios o tendencias por periodos.
- **Datos Libres:** La plataforma brinda los datos censados en tiempo real de manera gratuita, no en su totalidad, de unos sensores por módulos específico, los cuales se pueden observar accediendo a la plataforma en calidad de invitado.

6.2. Trabajo Futuro

Aquí se presenta propuestas de trabajos basados en la plataforma, con tal de realizar una funcionalidad completa de solución.

Rutas de Ocio

Según la utilización de la plataforma, si un usuario busca rutas por donde la contaminación tiene menores proporciones en determinados lugares, la plataforma le brindará la ruta más favorable según los parámetros límites totables por el usuario.

Estimaciones de Patrones

Con la información censada, aplicando Data Mining, será posible identificar patrones en el comportamiento de los cambios climáticos y tener un control, valores e indicadores de los datos de los sectores que tengan las condiciones que se puedan necesitar para algún objetivo específico.

Predicciones de anomalías

Con la información censada, aplicando algoritmos de Machine Learning, será posible realizar predicción de valores por temporadas o sectores por

donde se tiene sensores almacenando información, a parte de poder realizar un estudio de los datos y según el requerimiento de un cliente, brindarle rutas, o sectores que tengan las condiciones que dicho cliente esta solicitando.

6.3. Comentario Final

Además de cumplir los requerimientos de la plataforma, también se cumplió los objetivos de competencias académicas planteadas. La plataforma se desarrolló utilizando la metodología de desarrollo SCRUM, así como también, se hizo uso de tecnologías Open Source.

El desarrollo de la plataforma sirve para fomentar, concientizar y monitorizar la realidad climática de nuestro país, generando así con los datos ofrecidos que propongan nuevas soluciones para seguir combatiendo el incremento de la contaminación a nivel global usando la tecnología que está al alcance de todos.

Bibliografía

- [1] ClearBlade Inc. Clearblade. Link:<https://www.clearblade.com/solutions.html>. [Último acceso: 15-Mayo-2017].
- [2] Inc Hologram. Hologram. Link:<https://hologram.io/cloud/>. [Último acceso: 12-Mayo-2017].
- [3] Wikipedia. Model-view-controller. Link:<https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller>. [Último acceso: 10-October-2017].
- [4] Google Inc. Angularjs framework mvw. Link:<https://plus.google.com/+AngularJS/posts/aZNVhj355G2>. [Último acceso: 18-Junio-2017].
- [5] Software Testing Material. Agile scrum methodology. Link:<http://www.softwaretestingmaterial.com/agile-scrum-methodology/>. [Último acceso: 25-Junio-2017].
- [6] Nitin Naik. Choice of effective messaging protocols for iot systems: Mqtt, coap, amqp and http. In *2017 IEEE International Systems Engineering Symposium (ISSE)*, pages 1–7, 2017.
- [7] Numbeo. World database of pollution index. Link:<https://www.numbeo.com/pollution/rankings.jsp>. [Último acceso: 08-Setiembre-2017].
- [8] The United Nations. Food and agriculture organization of united nations. Link:<https://www.fao.org/faostat/en/#country/170>. [Último acceso: 08-Junio-2017].

- [9] World Air Quality. Real-time air quality index. Link:<http://aqicn.org/city/peru/lima/san-martin-de-porres/>. [Último acceso: 28-Mayo-2017].
- [10] SENAMHI; La república. Verano 2016: radiaciones uv llegarán a niveles extremos. Link:<http://larepublica.pe/turismo/rumbos-al-dia/728521-verano-2016-radiaciones-uv-llegaran-niveles-extremos>. [Último acceso: 05-Enero-2017].
- [11] Wilson Daniel Pinto Rios. Monitoreo de cultivos con redes de sensores xbee, arduino y dispositivos de medición de suelos. In *Ingenieria de Sistemas y Computacion*. Universidad Tecnológica de Pereira, 2015.
- [12] Fernando García Juan Carmona Bladimir J. Pérez, José M. Koo. Automatización, monitoreo y control remoto de un sistema de riego agrícola con código abierto. In *Twelfth LACCEI Latin American and Caribbean Conference for Engineering and Technology*, pages 22–24, 2014.
- [13] Ridvan Canbaz Ayberk Calpbini Yasin Kabalci, Ersan Kabalci. Design and implementation of a solar plant and irrigation system with remote monitoring and remote control infrastructures. In *Solar Energy* 139, page 506 – 517, 2016.
- [14] Chetan Solanki Ravi Tejawani, Girish Kumar. Remote monitoring for solar photovoltaic systems in rural application using gsm voice channel. In *Energy Procedia* 57, page 1526 – 1535, 2014.
- [15] Association of Modern Technologies Professionals. Software development methodologies. Link:<http://www.itinfo.am/eng/software-development-methodologies/>. [Último acceso: 29-Noviembre-2017].
- [16] Avante. Comparativa de metodologías ágiles vs tradicionales. Link:<http://www.avante.es/>

- comparativa-metodologias-agiles-vs-tradicionales/.
[Último acceso: 29-Noviembre-2017].
- [17] SCRUM Organization. Scrum. Link:<https://www.scrum.org/>.
[Último acceso: 02-Mayo-2017].
- [18] Youry Khmelevsky; Xitong Li; Stuart Madnick. Software development using agile and scrum in distributed teams. In *Systems Conference. Systems Conference (SysCon)*, 2017 Annual IEEE International, 2017.
- [19] M. Rizwan Jameel Qureshi. Agile software development methodology for medium and large projects. In *IET Software*. Institution of Engineering and Technology, 2012.
- [20] Rabie A. Ramadanb Adel Alkhalila. Iot data provenance implementation challenges. In *Procedia Computer Science*. College of Computer Science and Engineering, Hail University, 2017.
- [21] Mitaro Namiki Pornpat Paethong, Mikiko Sato. Low-power distributed nosql database for lot middleware. *IEEE Xplore*, 2016.
- [22] Veronica D'Souza Saurabh Zunke. Json vs xml: A comparative performance analysis of data exchange formats. In *Periodicals of Engineering and Natural Sciences*. Vishwakarma Institute of Information Technology, University of Pune, Maharashtra, 2014.
- [23] K.R. Ramkumar Gaurav Goyal, Karanjit Singh. A detailed analysis of data consistency concepts in data exchange formats (json & xml). In *IEEE Xplore*. Computer Science and Engineering, Chitkara University, Punjab, India, 2017.
- [24] M. Bhalerao Sharvari Rautmare. Mysql and nosql database comparison for iot application. In *2016 IEEE International Conference on Advances in Computer Applications (ICACA)*, 2016.
- [25] Mark Summerfield. Programming in go: Creating applications for the 21st century. In *ISBN 0-201-63361-2*. Addison-Wesley, 2012.

- [26] Denduang Pradubsuwun Anuchit Prasertsang. Formal verification of concurrency in go. In *13th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, 2016.
- [27] Gregor Richards; Sylvain Lebresne; Brian Burg; Jan Vitek. An analysis of the dynamic behavior of javascript programs. In *PLDI 10 Proceedings of the 31st ACM SIGPLAN Conference on Programming Language Design and Implementation*. Addison-Wesley, 2010.
- [28] Ralf S. Engelschall. EcmaScript 6. Link:<http://es6-features.org/#Constants>. [Último acceso: 02-Mayo-2017].
- [29] Google Inc. Angularjs framework. Link:<https://angularjs.org/>. [Último acceso: 02-Junio-2017].
- [30] Information Sciences Institute. Rfc-821 simple mail transfer protocol. Link:<http://web.archive.org/web/20081018175957/http://rfc.net:80/rfc821.html>, 1982. [Último acceso: 30-Mayo-2017].
- [31] Paolo Patierno Software Embedded Engineer. Mqtt & iot protocols comparison. In *Smart Home & smart Factory Systems*, 2014.
- [32] S. Bandyopadhyay and A. Bhattacharyya. Lightweight internet protocols for web enablement of sensors using constrained gateway devices in computing, networking and communications (icnc). In *Computing, Networking and Communications*, page 334–340, 2013.
- [33] HTTP2 organization. Http/2. Link:<https://http2.github.io/>. [Último acceso: 15-Noviembre-2017].
- [34] MQTT organization. Mqtt. Link:<http://mqtt.org/>. [Último acceso: 05-Noviembre-2017].
- [35] A. Valera H.-X. Tan D. Thangavel, X. Ma and C. K.-Y. Tan. Performance evaluation of mqtt and coap via a common middleware. In *Intelligent Sensors, Sensor Networks and Information Processing*, pages 1–6, 2014.

- [36] Carsten Bormann. Rfc 7252 constrained application protocol. Link: <http://coap.technology/>. [Último acceso: 16-Noviembre-2017].
- [37] OASIS corp. Amqp a business internet protocol. Link: <https://www.amqp.org/about/what>. [Último acceso: 17-Noviembre-2017].
- [38] Paridhika Kayal and Harry Perros. A comparison of iot application layer protocols through a smart parking implementation. In *Innovations in Clouds, Internet and Networks (ICIN)*, 2017.
- [39] Andreas Miaoudakis Amaury Van Bemten Ioannis Askoxylakis Ioannis Papaefstathiou† Sotirios Katsikeas, Konstantinos Fysarakis and Anargyros Plemenos. Lightweight & secure industrial iot communications via the mq telemetry transport protocol. In *Computers and Communications*, 2017.
- [40] Richard; Johnson Ralph; Vlissides John Gamma, Erich; Helm. Design patterns: Elements of reusable object-oriented software. In *ISBN 0-201-63361-2*. Addison-Wesley, 1995.
- [41] Adam Altar Dragos-Paul Pop. Designing an mvc model for rapid web application development. In *Elsevier*, 2013.
- [42] Ala Al-Fuqaha; Mohsen Guizani; Mehdi Mohammadi; Mohammed Aledhari; Moussa Ayyash. Internet of things: A survey on enabling technologies, protocols and applications. *IEEE Xplore*, 2015.
- [43] Tom H. Luan; Longxiang Gao; Zhi Li; Yang Xiang; Guiyi We; Limin Sun. Fog computing: Focusing on mobile users at the edges. ., 2016.
- [44] Mohammad Aazam; Eui-Nam Huh. Fog computing and smart gateway based communication for cloud of things. *IEEE Xplore*, pages 464–470, 2014.
- [45] Divya Shrungar J; Priya M P; Asha S M. Fog computing: Security in cloud environment. *Journal of Cloud Computing:Advances, Systems and Applications*, pages 1803–1807, 2015.

- [46] Mohammad Aazam; Eui-Nam Huh. Fog computing micro datacenter based dynamic resource estimation and pricing model for iot. *IEEE Xplore*, pages 687–694, 2015.
- [47] Wikipedia. Transport layer security. Link:https://es.wikipedia.org/wiki/Transport_Layer_Security. [Último acceso: 19-Noviembre-2017].
- [48] ISSL. What is ssl? Link:<http://info.ssl.com/article.aspx?id=10241>. [Último acceso: 19-Noviembre-2017].
- [49] Ubuntu. Ufw. Link:<https://help.ubuntu.com/community/UFW>. [Último acceso: 18-Noviembre-2017].
- [50] OpenBSD. Openssh. Link:<https://www.openssh.com/>. [Último acceso: 19-Noviembre-2017].

Apéndice A

Definiciones

En este apéndice se expondrá las deficiones de los acrónimos que utilizamos como tecnicismo los cuales a veces suelen ser confusos, por lo cual, se pretende dejar en claro cada uno de ellos debido a que conocer estos conceptos son de suma importancia para entender el presente trabajo.

A.1. Servicios Web

Un Servicio Web (Web Service) describe una forma estándar de integrar aplicaciones Web mediante el uso de XML, SOAP, WSDL y UDDI sobre los protocolos de la Internet. Un servicio web se encarga de compartir la lógica del negocio, los datos y los procesos.

- **Arquitectura de un Servicio Web basado en SOAP**

El servicio web provider envía un archivo WSDL a UDDI. El cliente se comunica con UDDI para averiguar quién es el proveedor de los datos que necesita y, a continuación, se pone en contacto con el proveedor de servicios mediante el protocolo SOAP. El proveedor de servicios valida la solicitud de servicio y envía datos estructurados en un archivo XML, utilizando el protocolo SOAP. Este archivo XML sería validado de nuevo por el solicitante del servicio utilizando un archivo XSD.

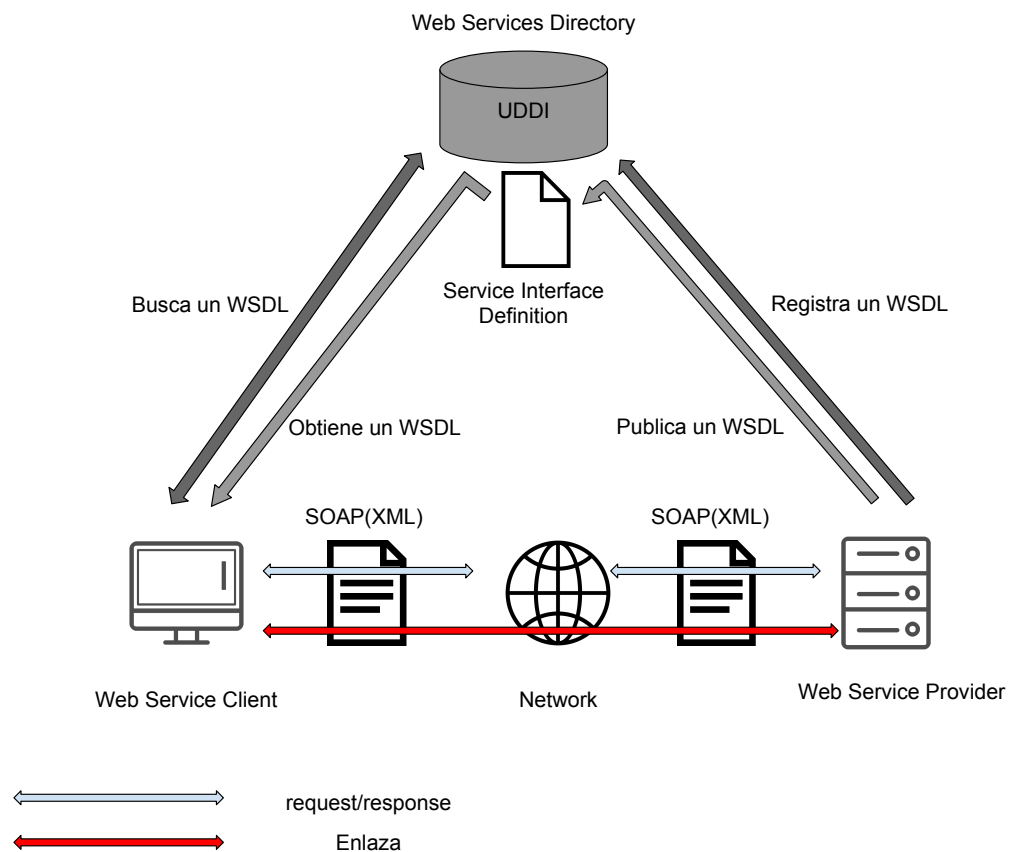


FIGURA A.1: Arquitectura de un Servicio Web basado en SOAP.
Fuente: Elaboración propia.

■ XML

Es un lenguaje de marcado que define un conjunto de parámetros y reglas codificando documentos en un formato legible por el usuario y por la máquina que envía y recibe información.

■ SOAP

Es un Protocolo para el intercambio de información estructurada en la implementación de servicios web en redes informáticas. Organiza la información utilizando XML y generalmente a través del protocolo HTTP para la transmisión de mensajes.

▼ Response headers	
Content-Length	2000
Content-Type	text/html; charset=utf-8
Date	Fri, 12 May 2017 15:51:02 GMT
Server	beegoServer:1.7.1
▼ Request headers	
Accept	text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Encoding	gzip, deflate
Accept-Language	en-US,en;q=0.5
Access-Control-Request-Headers	content-type
Access-Control-Request-Method	POST
Connection	keep-alive
Host	
Origin	http://beagons.uni.edu.pe
User-Agent	Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:53.0) Gecko/20100101 Firefox/53.0

FIGURA A.2: Estructura de un mensaje SOAP entre el cliente web y el servicio rest. Fuente: Elaboración propia.

■ UDDI

Es un protocolo que sirve para publicar la información de los servicios web. Permite comprobar qué servicios web están disponibles. Así que cuando un sistema de software necesita un informe y/o datos en particular, iría a la UDDI y averiguaría qué otro sistema puede contactar para recibir esos datos.

■ WSDL

Es el lenguaje de la interfaz pública para los servicios web. Es una descripción basada en XML de los requisitos funcionales necesarios para establecer una comunicación con los servicios web.

A.2. API Web

Una API Web (Web API) es un desarrollo en servicios web donde se ha hecho énfasis en las comunicaciones basadas en la transferencia de estado representacional (REST) más simples.

■ Arquitectura de un REST Web API

El servicio web ejecuta un servicio sobre HTTP, lo cual permite crear URLs que muestren información. El cliente accede a dicha información a través de un web browser, como si buscara una web, con un URLs ya definido puede obtener información que requiera usando HTTP. Se diferencia de SOAP en que REST opera sobre recursos, no sobre servicios.

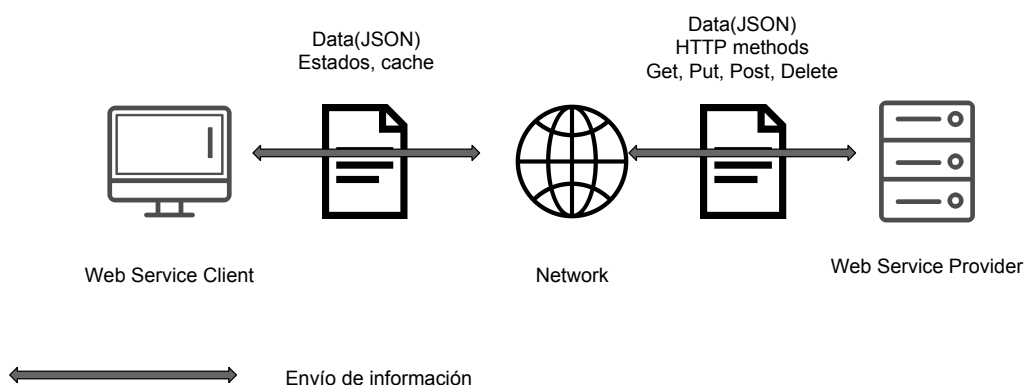


FIGURA A.3: Arquitectura de un Servicio Web basado en REST.
Fuente: Elaboración propia.

■ REST

En un RESTful API no requieren de protocolos basados en XML (SOAP y WSDL) para el intercambio de información, sino que está basado en formato JSON. REST provee de 4 métodos con el cual puede realizar interacciones con la información, estos son GET, POST, PUT, DELETE.

■ RADL

RESTful API Description Language es el lenguaje de la interfaz pública para los servicios web. Es una descripción basada en JSON de los requisitos funcionales necesarios para establecer una comunicación con los servicios web.

■ Route

Define los URL patterns, está conformado por los namespaces, seguido de los controllers, methods y finalmente los parameters.

■ Routing

Es el proceso de seleccionar una ruta para el tráfico en una red, o entre o entre múltiples redes. A nivel de REST, las rutas son los URLs y se utilizan para el intercambio de información.

■ Restful Routing

Es el encargado de resolver las consultas y devuelve responsive data, según los url patterns a los cuales se comunican mediante http.

■ URL Pattern

El URL pattern es la ruta que es evaluada por el routing y está ubicada justo después del domain name de la plataforma web.

En el caso del RESTful API, se presenta un ejemplo de un URL patterns:

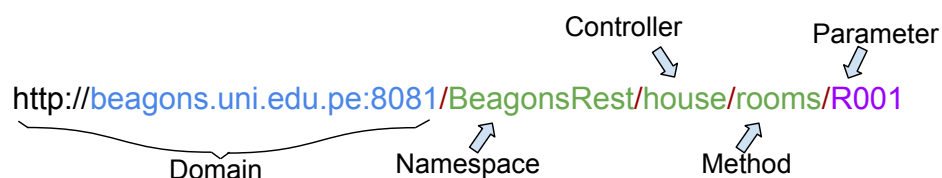


FIGURA A.4: Ejemplo de restful routing. Fuente: Elaboración propia.

A.3. Recursos Web

Los recursos web (Web Resources) es un termino utilizado en Arquitectura Web para referirse a los targets de los URI o IPs estáticos o no estáticos de documentos, files, entidades, etc.

■ URI

Es una cadena de caracteres que identifica los recursos de una red de forma unívoca. Puede ser un URL o un URN o ambos.

URL: Especifica la localización de algo, sigue el siguiente patrón:

$[Scheme] : // [Domain] : [Port] / [Path] ? [Querytring] \# [FragmentId]$

URN: Epecifica el nombre de algo, sigue el siguiente patrón:

$[namespaceId] / [Pathtofile]$

Como por ejemplo:

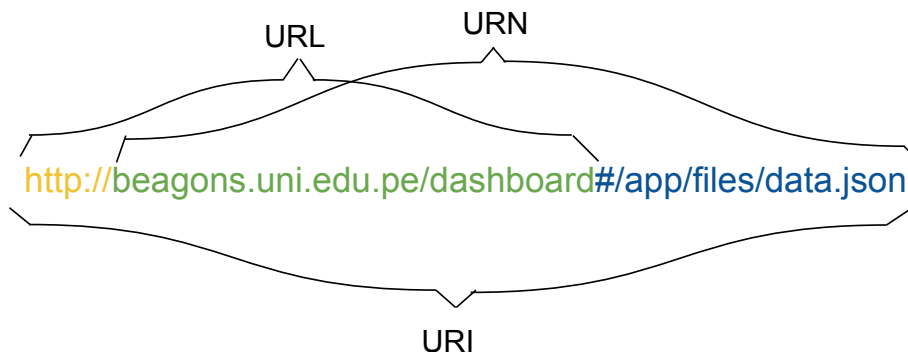


FIGURA A.5: Estructura de un URI. Fuente: Elaboración propia.

■ Dominio

Es el dominio (Domain) o nombre designado por un URL único que es interpretado por el web browser con el cual busca un Ip asociado y mostrar (en caso encontrar) un website.

■ **Hosting**

Es el lugar en donde se realiza el servicio, es decir, es el servidor. En el cual se almacena información de los usuarios, ya sea fotos, videos o cualquier contenido accesible vía web.

■ **Servidor Web**

Un servidor web (Web Server) es un computador que ejecuta una aplicación de software diseñada para atender peticiones y dar respuestas expresándoles a través de una página web.

■ **Página Web**

Una página web (Web Page) es un documento (puede ser HTML) diseñado para mostrarse en un web browser.

■ **Página Estática**

Una página estática (Landing Page) es una página web con características específicas, generalmente su utilización es para difundir información, generalmente se le denomina así al "index.html".

■ **Sitio Web**

Un Sitio Web (Web Site) está conformado de una o más WebPages, por ejemplo el sitio web puede ser <https://beagons.uni.edu.pe>, mientras que sus páginas serían <http://beagons.uni.edu.pe/dashboard>, <http://beagons.uni.edu.pe/contactUs>, etc.

■ **Aplicación Web**

Una aplicación web (Web Application) es un sitio web que tiene más funcionalidades que sólo mostrar información, se provee de interacción con el usuario tal como subir, editar información o archivos, mostrar estadísticas, etc.

Interactúa con el servidor web mediante realizadas por los usuarios. Esta estructurada en 3 capas:

Capa del navegador: Encargada de interpretar los lenguajes de marcado y de navegadores para interactuar con los usuarios, tales como HTML, XML y javascript (en su mayoría de casos).

Capa del servidor: Encargada de gestionar y manejar las peticiones y consultas hacia una base de datos u otro servicio, en nuestro caso, utilizamos un servicio RESTful.

Capa de persistencia: Encargada de almacenar los datos utilizando un DBMS, el cual contiene almacenado nuestra data, la cual será consultada por la capa del servidor o servicio RESTful para posteriormente ser mostrada en la capa del navegador o cliente web.

■ **Plataforma Web**

Una plataforma web (Web Platform) se conforma de una aplicación web, aplicaciones móviles y la comunicación entre este y otros servicios externos. Además de una integración más completa entre cada uno de los elementos que la conforman.

A.4. Seguridad en Redes

La seguridad en redes (Network Security) se definen como métodos o maneras para prevenir y vigilar el acceso no autorizado.

Los protocolos de seguridad utilizados son:

- **Transport Layer Security**

TLS [47] es un protocolo criptográfico que proporciona conexiones seguras por una red de Internet. Es una actualización de SSL 3.0 definido por el RFC 2246. Sirve para el intercambio de información encriptada entre un servidor y los clientes, se ejecuta sobre las capas TCP/UDP.

- **Secure Sockets Layer**

SSL [48] es un protocolo criptográfico que proporciona conexiones de seguridad en una red de Internet, fue desarrollado por primera vez por NetScape. Se usan certificados X.509, la cual es una criptografía asimétrica.

- **Uncomplicated FireWall**

UFW [49] es un firewall diseñado por Ubuntu. Mediante el CLI se puede configurar el tráfico ingoing, outgoing en puertos específicos y también que tipo de servicio puede enviar/recibir por medio de ese puerto, es decir, configura las iptables.

- **Secure SHell**

SSH [50] es un protocolo criptográfico que proporciona conexiones seguras por una red de Internet, sirve para acceder servidores privados a través de un CLI. Permitiendo manejar el servidor remoto por completo.

Apéndice B

Ingeniería del Software

En este apéndice se expone los diagramas de casos de uso y las especificaciones de software de cada actor.

B.1. Casos de Uso

Son una descripción de los pasos o actividades que deben realizarse para llevar a cabo un proceso. Las entidades que participan en un caso de uso se denominan actores.

Los diagramas de casos de uso sirven para especificar la comunicación y el comportamiento de un sistema mediante su interacción con los usuarios y/u otros sistemas.

B.1.1. Administrador

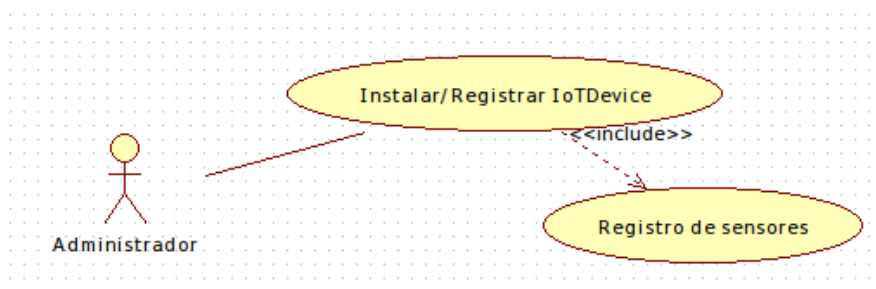


FIGURA B.1: Casos de uso del administrador del sistema. Fuente: Elaboración propia.

B.1.2. Usuario

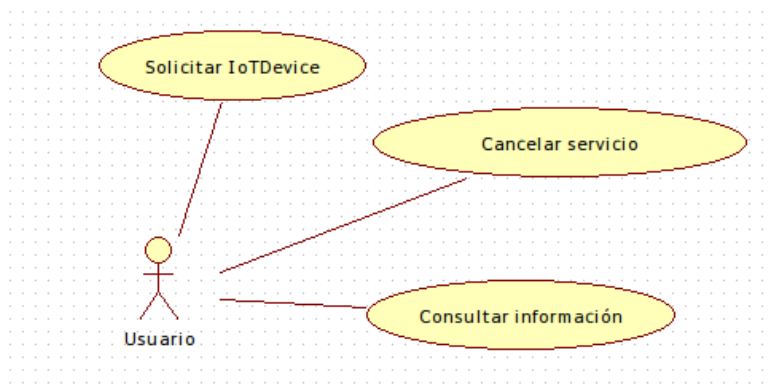


FIGURA B.2: Casos de uso del usuario. Fuente: Elaboración propia.

B.2. Especificaciones de los Casos de Uso

B.2.1. Caso de uso: Usuario

1. Breve descripción

Permite al cliente solicitar uno o más módulos al administrador, cancelar servicio y visualizar información de la data censada.

1.1. **Solicitar srvcio:** El cliente se comunica con Administrador de la plataforma para solicitar un servicio, indicando las ubicaciones y lista de sensores por cada módulo.

1.2. **Cancelar Servicio:** El cliente puede cancelar el servicio.

1.3. **Consulta Información:** El cliente puede ingresar a la plataforma web y puede ver la información que censa y luego es mostrado.

2. Pre-condiciones

Sistema de almacenaje habilitado. Sensores funcionando. Sistemas de servicios (web, RESTful) habilitados.

3. Flujo Básico

El caso de uso es inicializado cuando un cliente ya conversado con

el administrador, se procede a instalar y posteriormente a registrar el módulo, una vez realizado, registra al usuario con sus respectivos módulo, o también puede cancelar el servicio y además entrando al sistema puede ver los datos que los dispositivos censan.

3.1. Solicitar módulo

Actor	Sistema
1. El cliente solicita la cancelación del servicio y la justificación	2. El administrador recibe la solicitud junto a la justificación, procediendo a eliminar y desasociar los módulos asignados al usuario.

CUADRO B.1: Especificación de los Casos de uso del Usuario - Solicitar módulo

3.2. Cancelar Servicio:

Actor	Sistema
1. El cliente solicita la cancelación del servicio y la justificación	2. El administrador recibe la solicitud junto a la justificación, procediendo a eliminar y desasociar los módulos asignados al usuario.

CUADRO B.2: Especificación de los Casos de Uso del Usuario - Cancelar Servicio

3.3. Consultar Data Censada:

Actor	Sistema
1. El cliente interactúa con la plataforma	2. La plataforma muestra la data censada por cada dispositivo con los respectivos sensores asociados, así como también las ubicaciones en un mapa.

CUADRO B.3: Especificación de los Casos de Uso del Usuario - Consultar Data

B.2.2. Caso de uso: Administrador

1. Breve descripción

Permite al administrador registrar el módulo una vez ya instalado y establecido en el lugar correspondiente.

Instalar Módulo: Una vez recibida la solicitud del cliente para la lista de sensores y su posición en el mapa, se instala el dispositivo en las coordenadas especificadas.

Registrar Módulo: Una vez ya instalado el módulo, pasamos a registrarlo en la plataforma, el cual tendrá su posición en el mapa, junto a la lista de sensores asociados a el.

2. Pre-condiciones

Sistema de almacenaje habilitado. Sensores funcionando. Sistemas de servicios (web, restful) habilitados.

3. Flujo Básico

3.1 Registrar/Instalar Módulo:

Se inicializa cuando el administrador decide registrar los módulos de un cliente.

Actor	Sistema
1. El cliente se comunica con el administrador de la plataforma	2. El administrador recibe la solicitud del cliente y comienza la instalación de los módulos especificados.
3. El cliente realiza las respectivas acciones del proceso de solicitud.	4. El administrador luego de recibir la verificación de los pasos realizados por el cliente, registra a los módulos asociándolos al cliente.
	5. El administrador registra al cliente y luego le asigna un usuario y contraseña para que acceda a la plataforma.

CUADRO B.4: Especificación de los Casos de Uso del Administrador

Apéndice C

Desarrollo de Software Ágil

En este apéndice se describe la metodología utilizada para el desarrollo y sus respectivos pasos o sprints backlogs realizados.

C.1. SCRUM

El método de desarrollo de software utilizado es SCRUM, el cual se basa en entregas parciales, constantes y regulares del producto final priorizadas según el beneficio que genera.

SCRUM es una metodología dónde se necesita obtener resultados en la brevedad posible con variantes o poco definidos, donde la innovación, la competitividad, la flexibilidad y la productividad son fundamentales.

Características

- Desarrollo optimizado con entregas periódicas constantes.
- Relación cercana en colaboración con el cliente.
- Interacción directa y constante con temática de conversación los cambios en requisitos con el cliente.
- Fomentación de la colaboratividad y trabajo en equipo, fomentando también un entorno de comunicación y confianza.
- Motivación por la autogestión y libre seguimiento individual.

C.1.1. Componentes

Scrum tiene un conjunto de reglas y roles los cuales rigen cierto control sobre el desarrollo ágil del proyecto.

- **Roles Principales:** Se distribuye el trabajo en base a roles asignados.

Product Owner: Es el encargado de velar de que el equipo trabaje de forma adecuada basado en la perspectiva del producto. Se encarga de escribir las historias de usuario y las archiva en el Product BackLog.

ScrumMaster: Es el encargado de eliminar obstáculos que puedan interrumpir y obstaculizar el avance del objetivo de sprint. El ScrumMaster actúa como mediador entre el equipo y posibles distracciones, haciendo que se cumplan las reglas y plazos establecidos.

Equipo Scrum: El equipo tiene la responsabilidad de realizar las entregas del producto. Se recomienda un equipo de entre 5 y 9 personas con habilidades necesarias para realizar el trabajo (análisis, diseño, desarrollo, pruebas, documentación, etc).

- **Roles Auxiliares:** Se distribuye el trabajo en base a roles asignados.

Stakeholders: Son los encargados de hacer viable el proyecto, son quienes se beneficiarán del proyecto justificando su desarrollo. Participan exclusivamente en las revisiones de los Sprints.

Managers: Son los encargados de establecer y estandarizar el entorno de desarrollo para el proyecto.

- **Sprint:** Es el período en el cual se desarrolla los objetivos. Los Sprints se desarrollan en tiempo constante, los cuales son definidos por el equipo basándose en su experiencia como equipo de desarrollo con un tiempo mínimo de 2 semanas y máximo de 4.
- **Historia de Usuario:** Es una representación de un requisito (pruebas de validación, discusiones) escrito utilizando el lenguaje común del usuario. Permitiendo responder rápidamente a los requisitos cambiantes.

- **Documentos:** Son los documentos que deben reportarse.

Product BackLog: Es el documento generado por el Product Owner, va dirigido a todo el proyecto y contiene descripciones genéricas de las funcionalidades que se requieren, las cuales están priorizadas, contiene las estimaciones a largo plazo, tanto como el valor y el esfuerzo requerido, estos datos ayudan a poder ajustar una línea de tiempo de prioridades de tareas alcanzables.

Sprint BackLog: Es el documento que genera el ScrumMaster junto con el Product Owner basado en el Product Backlog, va dirigido al equipo Scrum y contiene un conjunto requisitos que deben ser desarrollados durante el siguiente sprint. Se define cómo será el desarrollo de los requisitos, dividiéndose en tareas con ciertas horas de trabajo asignadas por cada una teniendo en cuenta que ninguna tiene una duración mayor a 16 horas. Se establece que si una tarea amerita más de 16 horas, será dividida en subtareas. Así como también, las tareas nunca se asignan, el equipo las va tomando según lo vean conveniente.

C.1.2. Proceso

El proceso de desarrollo ágil es:

- **Toma de requisitos:** Se hace una lista de requisitos y se crea una historia de usuario para cada uno. Además se define la lógica que debe seguirse en el desarrollo.
- **Planear Sprint:** Se archiva todas historias de usuario en un Product BackLog y se planea un Sprint.
- **Realización del Sprint:** El ScrumMaster toma una porción del Product BackLog y el sprint planeado generando un Sprint BackLog, se realiza los planes y asignaciones del trabajo.
- **Revisión del Sprint:** Una vez terminado el sprint, se coordina una reunión con el Product Owner presentándole los resultados realizados

durante el sprint. Si se cumplió el requerimiento se planea el siguiente Sprint. En caso de que no, se vuelve a realizar el sprint con las recomendaciones y cambios acordados.

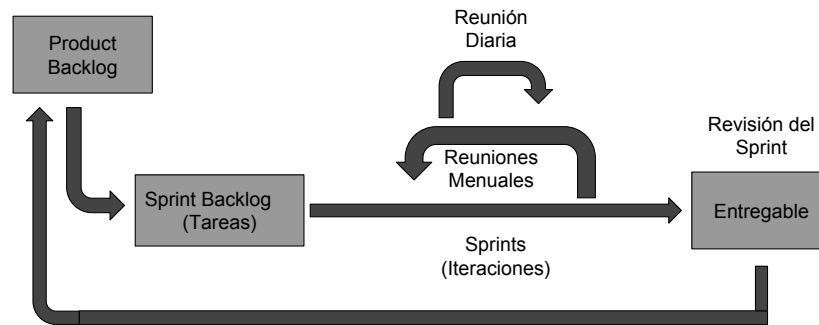


FIGURA C.1: Estrategia de desarrollo ágil SCRUM. Fuente: Software Testing Material. [5]

Este proceso se repite hasta finalizar el proyecto.

C.2. Sprint Backlogs

En esta sección se describen los Sprint Backlogs acordados para obtener un entregable periódicamente, el cual es desarrollado siguiendo las reglas de la metodología SCRUM, mostrando tiempo de duración, resultado de la revisión y observaciones realizadas.

C.2.1. Primer Sprint Backlog

Historia de Usuario	Especificación	días	horas/día
1. Diseño de la arquitectura del servicio web	Escoger que arquitectura de diseño adecuada para aplicar e implementar en la plataforma.	2	4
2. Selección de la herramientas de desarrollo	Escoger y realizar pruebas con las herramientas de desarrollo para aplicar el patrón de arquitectura escogida previamente.	4	5
3. Crear un prototipo web	Basado en la arquitectura de desarrollo y las herramientas, se implementó un prototipo que abarque y contenga las características mencionadas	4	5

CUADRO C.1: Primer Sprint Backlog

Resultados del Sprint:

- **Tiempo estimado:** 10 días
- **Estado:** Modificado
- **Observaciones:** Las herramientas y la arquitectura de desarrollo planteadas y escogidas para el desarrollo del primer prototipo eran rígidas y muy estrictas en su implementación, por lo que se optó por realizar otra investigación sobre los 3 puntos ya planteados.

C.2.2. Segundo Sprint Backlog

Historia de Usuario	Especificación	días	horas/día
1. Diseño de la arquitectura del servicio web	Modificación del Sprint Backlog anterior, se debe escoger una arquitectura con la cual el desarrollo sea más sencillo y sea flexible.	2	4
2. Selección de la herramientas de desarrollo	Modificación del Sprint Backlog anterior, se debe escoger y realizar pruebas con las nuevas herramientas de desarrollo planteadas.	4	5
3. Crear un prototipo web	Modificación del Sprint Backlog anterior, Utilizando las nuevas herramientas y arquitectura se implementó un prototipo	4	5

CUADRO C.2: Segundo Sprint Backlog

Resultados del Sprint:

- **Tiempo estimado:** 10 días
- **Estado:** Aceptado
- **Observaciones:** Ninguna.

C.2.3. Tercer Sprint Backlog

Historia de Usuario	Especificación	días	horas/día
1. Diseño y creación de la base de datos	Se debe diseñar una base de datos basado en los requisitos de flexibilidad de la plataforma al momento de generar consultas.	4	5
2. Inserción manual de información para pruebas	Se inserta información usando el CLI del DBSM.	4	5
3. Crear un prototipo de servicio REST	Se implementa un prototipo de servicio RESTful, con la base de datos ya modelada y creada anteriormente, para poder ser consumida mediante métodos GET y POST.	8	7

CUADRO C.3: Tercer Sprint Backlog

Resultados del Sprint:

- **Tiempo estimado:** 16 días
- **Estado:** Aceptado
- **Observaciones:** Con el prototipo actual, ya se puede pasar a una fase de desarrollo, además de mejorar ciertos aspectos en el diseño de la base de datos.

C.2.4. Cuarto Sprint Backlog

Historia de Usuario	Especificación	días	horas/día
1. Diseño y creación de la base de datos	Modificación del Sprint anterior, se mejora algunos aspectos del modelado de la base de datos.	2	5
2. Inserción manual de información para pruebas	Se vuelve a insertar información usando el CLI del DBSM con el nuevo modelo.	2	5
3. Crear un prototipo de servicio REST	Se implementa un prototipo de servicio RESTful, basado en el nuevo modelo.	3	6
3. Implementar un RESTful API	Utilizando el prototipo REST, se debe implementar un Restful API mediante el cual se pueda realizar consultar a la base de datos vía web recibiendo información.	6	5
4. Realizar pruebas de consultas al REST	Utilizando el prototipo web presentado anteriormente, se pretende manejar consultas al servicio REST desde el protitipo web.	6	5

CUADRO C.4: Cuarto Sprint Backlog

Resultados del Sprint:

- **Tiempo estimado:** 19 días
- **Estado:** Aceptado
- **Observaciones:** Ninguna.

C.2.5. Quinto Sprint Backlog

Historia de Usuario	Especificación	días	horas/día
1. Implementación de la página home de la plataforma web	Viendo los requerimientos del Product Owner, se diseña una pagina web de presentación.	4	8
2. Implementación de la interfaz de mensajería web	Viendo los requerimientos, se diseña e implementa una interfaz de mensajería con el cual interesados puedan contactar con los administradores.	4	8

CUADRO C.5: Quinto Sprint Backlog

Resultados del Sprint:

- **Tiempo estimado:** 8 días
- **Estado:** Aceptado
- **Observaciones:** Ninguna.

C.2.6. Sexto Sprint Backlog

Historia de Usuario	Especificación	días	horas/día
1. Implementación del dashboard de la plataforma web	Viendo los requerimientos, se diseña e implementa un dashboard que haga uso de servicio RESTful para obtener información.	14	8

CUADRO C.6: Sexto Sprint Backlog

Resultados del Sprint:

- **Tiempo estimado:** 14 días
- **Estado:** Modificado
- **Observaciones:** Algunas vistas de administración deben cambiarse.

C.2.7. Séptimo Sprint Backlog

Historia de Usuario	Especificación	días	horas/día
1. Implementación del dashboard de la plataforma web	Modificación del sprint anterior, se re diseña algunos factores de las vistas.	7	8
2. Implementación del login de la plataforma web	Por cuestiones de seguridad, se implementa un login que consulte al servicio RESTful para la autenticación de usuarios.	3	5
3. Implementación de la administración de usuarios	Se debe poder crear, eliminar y modificar usuarios del sistema.	5	5
4. Implementación de la administración de módulos	Se debe poder crear, eliminar y modificar módulos. Para todos los módulos se debe registrar sus ubicaciones, datos importantes, y sensores asociados.	5	4

CUADRO C.7: Séptimo Sprint Backlog

Resultados del Sprint:

- **Tiempo estimado:** 20 días
- **Estado:** Aceptado
- **Observaciones:** Añadir algunas funcionalidades como la ubicación de los módulos y mapas de calor.

C.2.8. Octavo Sprint Backlog

Historia de Usuario	Especificación	días	horas/día
1. Visualización de los datos censados	Los usuarios pueden visualizar los datos obtenidos en tiempo real mediante gráficas que muestra los sensores por cada módulo.	6	5
2. Visualización de un histórico de datos censados	Los usuarios pueden visualizar un histórico de los datos obtenidos, filtrando por fechas, horas y número de valores que quieran visualizar.	6	4

CUADRO C.8: Octavo Sprint Backlog

Resultados del Sprint:

- **Tiempo estimado:** 12 días
- **Estado:** Aceptado
- **Observaciones:** Ninguna.

C.2.9. Noveno Sprint Backlog

Historia de Usuario	Especificación	días	horas/día
1. Visualización de la ubicación de módulos	Los usuarios pueden visualizar en dónde se encuentran y cuánta área están abarcando para censar datos.	6	7

2. Visualización de mapas de calor	Los usuarios pueden visualizar mapas de calor proyectadas por las áreas cubiertas por cada módulo según el valor censado de cada tipo de sensor en ese instante.	6	6
------------------------------------	--	---	---

CUADRO C.9: Noveno Sprint Backlog

Resultados del Sprint:

- **Tiempo estimado:** 12 días
- **Estado:** Aceptado
- **Observaciones:** Ninguna.

C.2.10. Décimo Sprint Backlog

Historia de Usuario	Especificación	días	horas/día
1. Diseño de una arquitectura Cloud Computing de servicios web	Una vez ya implementado la plataforma web, se hace uso de tecnologías Cloud Computing para obtener un Ip pública y dominio para que se pueda acceder desde cualquier navegador web, levantando el servicio, configurando la base de datos, etc.	5	4

CUADRO C.10: Décimo Sprint Backlog

Resultados del Sprint:

- **Tiempo estimado:** 5 días
- **Estado:** Aceptado
- **Observaciones:** Ninguna.

C.2.11. Decimo Primer Sprint Backlog

Historia de Usuario	Especificación	días	horas/día
1. Monitorización de datos en tiempo real	La data obtenida debe ser mostrada y registrada al mismo tiempo que llega al servidor.	5	8

CUADRO C.11: Decimo Primer Sprint Backlog

Resultados del Sprint:

- **Tiempo estimado:** 5 días
- **Estado:** Aceptado
- **Observaciones:** Ninguna.

C.2.12. Decimo Segundo Sprint Backlog

Historia de Usuario	Especificación	días	horas/día
1. Diseño de una arquitectura Fog Computing	Una vez ya establecido el sistema y la plataforma, pasamos a optimizar los procesos ejecutados por el Cloud, esto se realiza a través de delegación, es decir, los servidores intermedio (o fog nodes) se encargan de reenviar la data para almacenar y mostrarla en tiempo real en la plataforma web.	8	5

CUADRO C.12: Decimo Segundo Sprint Backlog

Resultados del Sprint:

- **Tiempo estimado:** 8 días
- **Estado:** Aceptado
- **Observaciones:** Ninguna.

Tiempo de desarrollo total de la plataforma

Para el desarrollo del sistema se trabajó 4 días por semana, con 6 horas por día.

- **Tiempo en días:** 139 días.
- **Tiempo en horas:** 814 horas.
- **Tiempo en semanas:** 34 semanas (8 meses y medio) aproximadamente.

Apéndice D

Manual y Guías de instalación

En este apéndice se expondrá los software, librerías y demás cosas necesarias para poder tener nuestro servidor activo, el servidor Cloud Master es una máquina con sistema operativo Ubuntu 16.04 y los servidores Fog Nodes son raspberry pi 3 con sistema operativo Rasbian.

D.1. Servidor Cloud

En el servidor Cloud, denominado de esta forma debido a que el servidor Cloud gestiona y maneja todos los datos que se tienen y además de los servicios en general, se necesita las siguientes cosas:

D.1.1. Lenguaje de programación Go

El lenguaje de programación Go y además de ciertas librerías para la implementación, se instalan así:

Primero, descargas y descomprimes el código fuente de Golang:

```
wget https://storage.googleapis.com/golang/go1.7.linux-amd64.tar.gz
sudo tar -C /usr/local -xzf go1.7.linux-amd64.tar.gz
tar -C ./ -xzf go1.7.linux-amd64.tar.gz
```

Luego en el fichero /etc/profile, añade la línea:

```
export PATH=PATH:/usr/local/go/bin
```

Y finalmente, en el fichero /home/\$USER/.profile añadir:

```
export GOPATH=$HOME/golangProjects
export GOROOT=$HOME/go
export PATH=$PATH:$GOROOT/bin
```

Donde golangProjects es el directorio donde estarán todos los proyectos de Go.

BeeGo

Es un RESTful Framework, diseñado para soportar intercambio de datos mediante HTTP.

```
go get github.com/astaxie/beego
go get github.com/beego/bee
```

GoMail

Es el driver del protocolo SMTP para Golang, el cual provee un conjunto de librerías y métodos para poder enviar mails o recibirlos.

```
go get gopkg.in/gomail.v2
```

Paho MQTT Golang

Es el driver del protocolo MQTT para Golang, contiene métodos y librerías para realizar la conexión a un broker y gestionar los mensajes que circular en dicho protocolo.

```
go get github.com/eclipse/paho.mqtt.
```

Mgo

Es el driver para MongoDB del lenguaje Golang, provee un conjunto de métodos para la manipulación y conexión a una base de datos.

```
go get gopkg.in/mgo.v2
go get gopkg.in/check.v1
```

D.2. Servidores Fog

En los servidores Fog, denominado de esta forma debido a que un servidor Fog gestiona y maneja datos por sectores asignados y específicos además de los servicios locales, se necesita las siguientes cosas:

D.2.1. Lenguaje de programación Python

Viene por defecto en el sistema operativo Raspbian.

PIP

Python Package Index es un Package Manager System (Sistema de Gestión de Paquetes) que se utiliza para instalar y administrar paquetes de software escrito en y para el lenguaje Python.

```
sudo apt-get install python-pip
```

Paho MQTT Python

Es el driver del protocolo MQTT para Python, contiene métodos y librerías para realizar la conexión a un broker y gestionar los mensajes que circular en dicho protocolo.

```
sudo pip install paho-mqtt
```

PyMongo

Es el driver para MongoDB del lenguaje Python, provee un conjunto de métodos para la manipulación y conexión a una base de datos.

```
sudo pip install pymongo
```

Coloredlogs

Es una librería para python que nos permite realizar logging con colores para distinguir entre avisos como info, error, tracer, warning, etc.

```
sudo pip install coloredlogs
```