EXTENSIÓN DE LA PLATAFORMA DE FUENTE ABIERTA THINGSBOARD PARA EL DESARROLLO DE SOLUCIONES IOT PARA EL AGRO COLOMBIANO

GERMÁN ANDRÉS LÓPEZ PACHECO

CRISTIAN FERNANDO MENDIVELSO SANABRIA

CARLOS ALBERTO RAMÍREZ OTERO

DIRECTORES

HÉCTOR FABIO CADAVID RENGIFO

WILMER EDICSON GARZÓN ALFONSO

ESCUELA COLOMBIANA DE INGENIERÍA JULIO GARAVITO

INGENIERÍA DE SISTEMAS

PROYECTO DE GRADO 2

BOGOTÁ

2018

Resumen

En este documento se detalla la extensión realizada a la plataforma de código abierto Thingsboard, con conceptos propios del agro colombiano (finca, lote y cultivo) para solucionar problemas que afectan la productividad de los cultivos, haciendo uso de herramientas de análisis de datos (Apache Spark) y aprendizaje automático (MLIB) en tiempo real y con datos históricos, con el fin de poder preveer posibles riesgos y amenazas que puede afectar el estado y crecimiento de los cultivos. Dentro de la implementación, se encuentra el uso de herramientas de georreferenciación que permiten ubicar regiones que representan los terrenos de los agricultores.

A lo largo del documento se encontrará más a detalle cada uno de los aspectos del proyecto con sus respectivos manuales para que pueda ser replicado y extendido en un futuro.

Contenido

1. Contexto (Proyecto)

1.1 Planteamiento del problema.

Colombia es un país con un enorme potencial en agricultura, gracias a su tamaño y diversidad geográfica. Desafortunadamente, está lejos de aprovecharla eficientemente: el 65% de sus tierras no están siendo utilizadas o no son aprovechadas debido bien sea por problemas políticos o por el uso de prácticas agrícolas obsoletas. A comparación de otros países, el nivel de productividad en la agricultura es bajo debido a que no se hace uso de herramientas tecnológicas.

1.2 Marco teórico y estado del arte.

1.2.1 Marco teórico

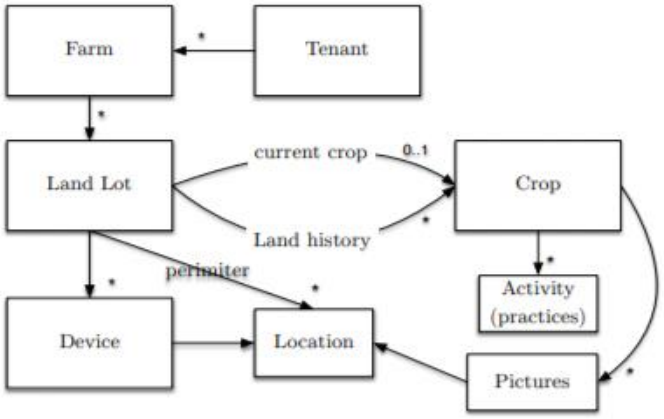
A partir de los avances tecnológicos tales como la teledetección (RS), Internet de las cosas (IoT), Big Data / Analítica de datos y los sistemas de información geográfica (GIS), ofrecen oportunidades que permiten contribuir a la productividad del agro, ya que ayudan a la mejor toma de decisiones. Estas tecnologías permiten no solo recopilar y analizar datos directamente de los cultivos en tiempo real, sino también extraer nuevos conocimientos del mismo. Además, este nuevo conocimiento, combinado con el conocimiento de expertos locales, podría convertirse en el núcleo de futuras herramientas de asistencia técnica y sistemas de apoyo a la decisión para países con una gran variedad de suelos tropicales y suelos como Colombia. [1]

1.2.2 Estado del arte

Dentro del proceso para iniciar con el desarrollo del proyecto, se revisaron escenarios de ciudades inteligentes [3] basados en la aplicación de nuevas tendencias tecnológicas como lo son el internet de las cosas y Big Data Analytics. Estamos en una sociedad donde han incrementado el número de dispositivos IoT y sus capacidades para manejar la información han avanzado muy rápido, pero el hecho de tener mucha información no implica que ya se pueda usar para generar conocimiento, por lo que se adaptan sistemas altamente capaces de clasificar y filtrar los datos generando los resultados que si marcan el factor diferencial para tomar decisiones efectivas.

Ya establecido un marco referente a las necesidades que presenta el agro colombiano, funcionalidades como el manejo de datos geoespaciales, información que reportan dispositivos como lo son sensores o drones. Han sido preparadas para su desarrollo dentro de la aplicación como extensión para el proyecto, y por lo que ha sido necesario la investigación particular por cada miembro del equipo en distintos temas encaminados al mismo objetivo, dentro del cual se puede destacar el uso de la analítica de datos usando herramientas como Spark, que además de ser de fuente abierta, se ha realizado una aplicación con alta posibilidad de extensión para diferentes escenarios. El proyecto ha sido orientado principalmente hacia los cultivos de papa, pero el desarrollo tiene el potencial de aplicarse a cualquier otro tipo de cultivo y además de incluir otros tipos de escenarios basados en la arquitectura propuesta (Detallada en el capítulo 3).

Para la selección de la herramienta, que ya posee las funciones necesarias para iniciar con el proceso de implementación a las necesidades del proyecto, se tuvo en cuenta herramientas de software libre y así mismo la selección estuvo basada en las capacidades y funcionalidades que ya poseían y que servirían para futuras adaptaciones a nuestras necesidades como también la facilidad de extensión que podíamos brindar; no solo se ha presentado la dificultad en comprender los estándares con los que se ha programado Thingsboard sino como adaptar otras soluciones como los mapas de Google, o APIs que manejan información climática y de terreno dentro de la solución de software, incluyendo la implementación de MongoDB para el soporte de análisis e indexación de datos geoespaciales y archivos, ya que para la base de datos NoSQL “Cassandra” que actualmente tiene implementado Thingsboard no era conveniente utilizar por las barreras en su ejecución que impedían realizar un avance significativo para el desarrollo del mismo.



Considerando que a partir de una finca, se posea un conjunto de lotes en el que se puede sembrar y cosechar un tipo diferente de cultivo establece un nivel de jerarquía que permite ampliar el número de usos como también la información que maneja, recordando que se espera un sistema con altas capacidades de generar conocimiento para toma de decisiones más acertadas en beneficio a la productividad de la agricultura, reducir el margen de problemas que ocurren normalmente en el crecimiento de cultivos y ejecutar acciones efectivas dependiendo de condiciones que estén reportando los cultivos y que para el agricultor puede ser un trabajo agotador y más cuando posee más de 1 cultivo.

Por medio de la investigación sobre analítica de datos en escenarios como Smart-cities, en el que permite entender su importancia y el funcionamiento en entornos donde la información está distribuida en silos independientes y que se pierden muchas oportunidades de mejora a la situación que presentan los entornos aplicados al sistema.
Con base a la toma de decisiones sobre datos recolectados en tiempo real, como también el histórico de datos, ha permitido generar acciones estratégicas para la optimización de recursos como también de los procesos en torno a contribuir con un sistema conjunto en beneficio a todos los participantes del mismo.

Selección de Solución de Software

Dentro de la solución de software que permite la implementación de varios de los objetivos que se tienen en cuenta para el desarrollo del proyecto se tuvieron 2 opciones: Thingsboard y FarmOs, ambos open-source y cuyo lenguaje de desarrollo son AngularJs-Java y Php respectivamente.

Para su selección se tuvieron en cuenta algunos aspectos fundamentales:

1. Dominio sobre el lenguaje de programación en el que está desarrollado
2. Modularidad de sus funciones para reutilización

A partir de los cuales Thingsboard fue la seleccionada ya que hay un fuerte conocimiento dentro de los lenguajes de programación en el que está implementado junto con la abstracción de los módulos necesarios para el manejo de datos a partir de sensores, y plugins para mejorar la experiencia de usuario, además de que permite expandir el concepto de herencia sobre su clase Asset lo cual amplía una mayor contribución sobre distintos casos de uso a los que se desee enfocar.

Plataforma para analítica de datos:

El procesamiento de los datos que se planteó va orientado al análisis en tiempo real, para esto se tuvieron en cuenta dos posibles herramientas, Apache Spark y Apache Storm.

Los datos a analizar son los generados por los sensores, que en Thingsboard se denominan datos de telemetría (temperatura, humedad, etc.)

Apache Spark puede integrarse con una herramienta llamada Spark Streaming garantiza la lectura de datos exactamente 1 vez en tiempos de micro-batching (casi tiempo real), mientras que Apache Storm garantiza la lectura de datos al menos una vez.

Apache Spark utiliza una estructura de datos llamada RDD ("Resilient Distributed Datasets"), los cuales utilizan un almacenamiento de datos en memoria y de una manera inmutable, lo que facilita su paralelización a través de un entorno clúster con facilidad, permitiendo así, cálculos en paralelo, y una arquitectura escalable [4].

Las operaciones sobre los RDD, se realizan de manera paralela sobre el clúster, existen 2 operaciones fundamentales map y reduce.

Cuando hablamos de Map i.e. RDD.map(func):

Lo que se pretende es que en el método map entre una operación func, y esta operación es aplicada a cada elemento del RDD retornando otro RDD resultante.

Cuando hablamos de Reduce i.e. RDD.reduce(func):

Lo que se pretende es que entre una operación func, pero en este caso ya no retorna otro RDD sino un valor único (single value).

La elección de Apache Spark como herramienta de analítica de datos en tiempo real, se dió gracias a su arquitectura previamente explicada y a que Thingsboard posee extensiones que soporta el uso de Spark y facilitan implementaciones dentro de este, además Apache Spark no solo cuenta con una comunidad amplia, sino que además posee librerías de machine learning, grafos y consultas sql.

Por último, tenemos la persistencia que maneja Thingsboard el cual almacena su información en la base de datos NoSQL Cassandra, pero dentro de las contribuciones propuestas, hacer el almacenamiento de archivos en grandes cantidades y la implementación de mapas para definir territorios dentro de las mismas se dificulta en esta base de datos:

1. Para el manejo de consultas geoespaciales hace uso de una aplicación llamada Apache Solr, pero su limitación es que no soporta índice esférico para inserción de polígonos. [5]
2. Cassandra puede manejar un sistema de partición de archivos, pero cada uno deberá ir en una tabla dividido en “Chunks” y por el cual dificulta un manejo de nombres y cantidad de tablas dentro de su esquema.

Por los cual algunas opciones son: Aerospike y MongoDB por lo cual MongoDB es otra base de datos NoSQL y posee índices Geoespaciales para rápida consulta y almacenamiento de estos objetos como también GridFS para el almacenamiento de archivos que exceden los 16MB y su partición en fragmentos permite una ágil recuperación de estos en consulta. [6]
Al igual que para el caso de Aerospike, este no presenta una documentación clara para el manejo de archivos, pero su soporte en georreferencia es similar al de MongoDB. [7]

Actualmente se han implementado pruebas de concepto entorno al procesamiento de datos de Spark simulando estos para generar alertas sobre Thingsboard como también la adaptación de mapas para el manejo del posicionamiento geoespacial del terreno para validar información suministrada sobre los lotes además de utilizar API’s para contribuir con información como lo es la climatología local.

1.3 Objetivos del proyecto: General y específicos

1.3.1 Objetivo General:

Adaptar la plataforma de software libre "Thingsboard" para satisfacer parte de los requerimientos planteados en el proyecto de Investigación de la Convocatoria Interna de la Escuela "Sistema de captura y procesamiento automático de datos para la caracterización de cultivos de papa para la región Cundiboyacense".

1.3.2 Objetivos Específicos

1.3.2.1 Extender la plataforma "Thingsboard" para que soporte conceptos propios utilizados en la gestión del agro en Colombia.

1.3.2.2 Agregar a la plataforma "Thingsboard" capacidades de indexación geo-referenciada y de manejo de imágenes multiespectrales para su uso en analítica de datos.

1.3.2.3 Extender las capacidades de procesamiento de datos en tiempo real de la plataforma "Thingsboard" de manera que se puedan integrar en la misma módulos de

alerta temprana basados en reglas predeterminadas, o clasificadores previamente entrenados.

1.4 Justificación

Gracias a las tendencias tecnológicas que han surgido en los últimos años tales como el internet de las cosas, Big Data y la analítica de datos, se han generado nuevas oportunidades para que a través de herramientas de TI se pueda contribuir a la mejora la productividad agrícola del país.
La propuesta es, que a través de un sistema que utilice las herramientas mencionadas anteriormente, y que esté adecuado al contexto del agro colombiano, pueda ser soporte para la toma de decisiones, ayude a la prevención de enfermedades en los cultivos y a través de un análisis de datos tanto en tiempo real como basado en históricos.

1.5 Área de aplicación del producto resultado del proyecto.

Este proyecto de grado contribuye con el desarrollo de un proyecto de la Escuela Colombiana de Ingeniería. Inicialmente su enfoque es académico debido al proceso de investigación necesario para entender el comportamiento de los cultivos a los que se centrará (Altiplano Cundiboyacense) y su aplicación va dirigido al sector agrícola en Colombia.

1.6 Cronograma de actividades que se observó

1.6.1 Cronograma Actividades Cristian Mendivelso

Semana	Lunes	Martes	Miércoles	Jueves	Viernes	Sábado	Domingo
1			(30 de mayo) Editar todos los checklist de la plataforma para que tengan un punto único de cambio (2 horas), Establecer una lista de chequeo de buenas prácticas para el cultivo (5	(31 de mayo) Establecer una lista de chequeo de buenas prácticas para el cultivo (1 hora), Extender el modelo para que el lote tenga una lista de dispositivos (3 horas), Cuando el	(1 de junio) Crear método que retorne informac ión de que tipo de cultivo y todos los datos de telemetr ía asociad os a	(2 de junio) Mostrar informac ión del cultivo y los datos de telemetrí a asociado s a esa fecha promedi ados (4 horas)	(3 de junio)

			horas)	dispositivo sea asociado a un lote, este lo guarde en el listado (3 horas)	cierta fecha (6 horas)		
2	(4 de junio)	(5 de junio)	(6 de junio) Cada punto que representa al dispositivo se tornará de un color distinto (verde y rojo) dependiendo del de la alarma (5 horas), Cuando haya una alarma de tipo Spark en el lote el lote cambia de color (3 horas)	(7 de junio)	(8 de junio) Cuando haya una alarma de tipo Spark en el lote el lote cambia de color (1 hora), Crear backend para enviar fotos al frontend (4 horas)	(9 de junio)	(10 de junio) Crear backend para enviar fotos al frontend (3 horas), Crear backend para retornar las foto georeferenciadas por fecha (3 horas)
3	(11 de junio)	(12 de junio) Crear un mapa que se ubique cerca a la ubicación del lote para el etiquetado(5 horas)	(13 de junio)	(14 de junio) Mostrar sobre el mapa todas las fotos georeferenciadas ubicadas correctamente (6 horas)	(15 de junio) Mostrar sobre el mapa todas las fotos georeferenciadas ubicadas correctamente (6 horas)	(16 de junio) Mostrar sobre el mapa todas las fotos georeferenciadas ubicadas correctamente (6 horas)	(17 de junio)

4	(18 de junio) Pruebas con Spark, utilizando Threads (6 horas)	(19 de julio)	(20 de julio) Adecuación de Apache Spark para que se verifiquen múltiples reglas al tiempo y no secuencialmente (6 horas)	(21 de julio)	(22 de julio)	(23 de julio) Adecuación código de Spark a RDD (6 horas)	(24 de julio) Adecuación código de Spark a RDD (6 horas)
5	(25 de junio)	(26 de junio) Instalación Clúster Spark y prueba de las aplicaciones corriendo (6 horas)	(27 de junio) Instalación Clúster Spark y prueba de las aplicaciones corriendo-corrección de errores (5 horas)	(28 de junio)	(29 de junio) Configuración Código de Spark para ser corrido en modo clúster (2 horas)	(30 de junio)	(1 de julio) Documentación Spark (7 horas)
6	(2 de julio) Edición esquema de reglas en Spark y acciones (6 horas)	(3 de julio) Hablar con mis compañeros en cómo repartirnos las tareas para avanzar con el libro de PGR, adelantar cosas que ya teníamos (4 horas)	(4 de julio) hacer correcciones pedidas del profesor Alexander en la interfaz (7 horas)	(5 de julio) Inicio documentación manual de Usuario Thingsboard (5 horas)	(6 de julio)	(7 de julio) actualizar última versión de Thingsboard en el servidor (5 horas)	(8 de julio) Documentación Manual de Usuario (4 horas)

7	(9 de julio)	(10 de julio) Realización de la cartelera para la vitrina (3 horas)	(11 de julio) Manual de Usuario Thingsboard, y edición del poster (6 horas)	(12 de julio) Integración del modelo de clasificación de árbol de clasificación con el conjunto de reglas de Spark, terminación del póster para la vitrina (7 horas)	(13 de julio) Trabajar en modelo de reconocimiento de imágenes, tratando de sacar un pedazo de la imagen por el polígono hecho por el usuario. Envío de datos desde el invernadero hasta los servidores donde está la plataforma Thingsboard (6 horas)	(14 de julio) Manual para implementación de nuevas reglas en apache Spark, planeación de la prueba de concepto para vitrina (3 horas)	(15 de julio)

8	(16 de julio)	(17 de julio) Realización de vídeo informativo de la herramienta Thingsboard para mostrar en vitrina (4 horas)	(18 de julio) Montaje de vitrina en la escuela (3 horas), realización de diapositivas para presentación de vitrina (2 horas)	(19 de julio) Vitrina académica (6 horas)	(20 de julio)	(21 de julio) Recursos (RESTful API) Landlots (3 horas)	(22 de julio) Vista dinámica (4 horas), Detalles finales libro (4 horas)

Tabla final:

	Fecha	Horas
1	30/05 - 05/06	24
2	06/06 -12/06	24
3	13/06-19/06	24
4	20/06-26/06	24
5	27/06-03/07	24
6	04/07-10/07	24
7	11/07-17/07	26
8	18/07-24/07	22

1.6.2 Cronograma Actividades Germán López

Semana	Lunes	Martes	Miércoles	Jueves	Viernes	Sábado	Domingo
1			(30 de mayo) Agregar barra de tiempo en front-end de Thingsboard, en la parte de lote (3 horas), explicar funcionamiento de mapas	(31 de mayo) Cuadrar con el estudiante de electrónica el tema de sensores (2 horas), Mapas ya cargar sin necesidad de	(1 de junio) Agregar marcas en la barra de tiempo para poder distinguir los días que se tomaron fotos	(2 de junio) Los polígonos de los lotes se pueden visualizar en los polígonos de la finca en el dashboard(5	(3 de junio) Corregir errores pequeños de front-end y cuando se borra un dispositivo o un lote en Thingsboard

			(1 hora)	botón (2 horas)	(3 horas), Dispositivos de tipo Spark se guardan en colección SparkDevice en Mongo(3 horas)	horas)	también se borra en MongoDB (2 horas)
2	(4 de junio) Cambiar formato de los polígonos para que se pueda hacer la georeferencia en Mongo (2 horas)	(5 de junio) Los puntos que representan los dispositivos en los lotes dentro de las fincas en el dashboard, al momento de presionarlos, muestran su último valor de telemetría (5 horas)	(6 de junio) Intentar cargar una imagen desde front-end hasta back-end (9 horas)	(7 de junio)	(8 de junio) Imagen pasa de front-end a back-end, pero no deja agregar una nueva finca (2 horas)	(9 de junio) La imagen ya pasa de front-end a back-end, sin perjudicar la finca (2 horas), Cargar una imagen desde Mongo hasta el front-end iniciativa (4 horas)	(10 de junio) Se pueden cargar y guardar cualquier tipo de imágenes (7 horas)
3	(11 de junio) Enviar varias imágenes desde el front-end hasta el back-end (5 horas)	(12 de junio) Las imágenes tienen metadatos (9 horas)	(13 de junio) La barra de tiempo poner marcas, dependiendo de las fechas que tienen las fotos (3 horas), Las imágenes se colocan sobre el mapa (3 horas)	(14 de junio) Hablar con estudiante de electrónica, para preparar los sensores del invernadero (2 horas)	(15 de junio) Dibujar polígono sobre imagen (2 horas). Enviar varios tipos de imágenes de front-end a back-end y viceversa y cambiar tipo de imágenes sobre el mapa (5 horas)	(16 de junio)	(17 de junio) Averiguar cómo paralelizar tareas en Spark (3 horas)
4	(18 de junio) Pruebas con Spark, utilizando Threads (4 horas)	(19 de julio)	(20 de julio)	(21 de julio)	(22 de julio)	(23 de julio)	(24 de julio)

5	(25 de junio) Averiguar sobre ejemplos y funcionamiento de ML en WEKA(5 horas)	(26 de junio) Entrenamiento de modelo de clasificación en WEKA y con el modelo ya entrenado se pudo guardarlo y luego cargarlo(7 horas)	(27 de junio) Averiguar modelo de clasificación en Spark, cómo entrenarlo y como persistirlo (7 horas)	(28 de julio) Realizar varias pruebas con el modelo árbol de decisión con un conjunto de datos hecho por mi y comparar varios resultados (4 horas)	(29 de julio) Realizar pruebas con modelos de regresión entrenándolos con el mismo conjunto de datos con que se entrenaron los modelos de clasificación para saber cuál es el mejor (4 horas)	(30 de julio) Buscar y comparar varios modelos de clasificación con la librería "mllib" de Spark para compararlos entre sí y ver cuál es el más efectivo (4 horas)	(1 de julio) Revisar varias fuentes sobre el funcionamiento del árbol de decisión desde el punto teórico y revisar con la documentación de Spark para saber cómo cambiar las características de este algoritmo (5 horas)
6	(2 de julio) Revisar con estudiante de electrónica el envío de sensores reales a la plataforma Thingsboard que reside en nuestro servidor (4 horas)	(3 de julio) Hablar con mis compañeros en cómo repartirnos las tareas para avanzar con el libro de PGR, adelantar cosas que ya teníamos (2 horas)	(4 de julio) Realizar diagrama de clases con los conceptos más representativos de la extensión hecha a Thingsboard, hacer correcciones pedidas del profesor Alexander en la interfaz (4 horas)	(5 de julio) Realizar correcciones de detalles en la parte de cultivos del lote, cambiar tamaños de mapas, los sensores en el dashboard muestran más de un dato de telemetría (7 horas)	(6 de julio) Avanzar con el libro haciendo los diagramas de secuencia que detallan los métodos más relevantes del proyecto (4 horas)	(7 de julio) Consultar papers o ensayos sobre experimentos donde comparan modelos de clasificación y actualizar última versión de Thingsboard en el servidor (5 horas)	(8 de julio) Consultar papers para poder tener una mejor base de la selección del modelo de clasificación y realizar requerimientos funcionales y no funcionales del software, los usuarios y glosario de términos de los conceptos aplicados al proyecto (3 horas)

7	(9 de julio) Realización de diagramas de despliegue y de actividades, primeros pasos para prueba de concepto (2 horas)	(10 de julio) Realización de la cartelera para la vitrina, buscar formas para que el modelo de clasificación, además de predicción también de la probabilidad (4 horas)	(11 de julio) Se encontraron 2 modelos de clasificación de la librería de mllib, que permiten dar probabilidad por cada predicción, pero el porcentaje de precisión baja. Avanzar con el libro se hizo el modelo de Cassandra (6 horas)	(12 de julio) Integración del modelo de clasificación de árbol de clasificación con el conjunto de reglas de Spark, terminación del póster para la vitrina (7 horas)	(13 de julio) Trabajar en modelo de reconocimiento de imágenes, tratando de sacar un pedazo de la imagen por el polígono hecho por el usuario. Envío de datos desde el invernadero hasta los servidores donde está la plataforma Thingsboard (6 horas)	(14 de julio) Detallar el funcionamiento original de Thingsboard, planeación de la prueba de concepto para vitrina (3 horas)	(15 de julio) Manual técnico sobre cómo agregar nuevos conceptos al Thingsboard, también se incluyó cómo agregar mapas a la plataforma (4 horas)
8	(16 de julio) Realizar primera prueba de concepto con sensores reales en la plataforma Thingsboard (4 horas), iniciativa de preparación de video para vitrina (3 horas)	(17 de julio) Realización de vídeo informativo de la herramienta Thingsboard para mostrar en vitrina (4 horas)	(18 de julio) Montaje de vitrina en la escuela (3 horas), realización de diapositivas para presentación de vitrina (2 horas)	(19 de julio) Vitrina académica (6 horas)	(20 de julio) Manual técnico del clasificador machine learning y establecer diagramas en el libro (3 horas)		
9							

Tabla final

	Fecha	Horas
1	30/05 - 05/06	28
2	06/06 -12/06	38
3	13/06-19/06	22
4	20/06-26/06	12
5	27/06-03/07	30
6	04/07-10/07	29
7	11/07-17/07	37
8	18/07-24/07	19

1.6.3 Cronograma Actividades Carlos Alberto Ramírez Otero

Semana	Lunes	Martes	Miércoles	Jueves	Viernes	Sábado	Domingo
1			(30 de mayo) Cargar la foto de la fachada de cada finca(6 horas)	(31 de mayo) Mostrar la foto como información general en los detalles de la finca (7 horas)	(1 de junio) Modificar el método que retorne información de que tipo de cultivo y todos los datos de telemetría asociados a cierta fecha (6 horas)	(2 de junio) Mostrar información del cultivo y los datos de telemetría asociados a esa fecha promediados (4 horas)	(3 de junio)
2	(4 de junio)	(5 de junio)	(6 de junio) Arreglar tipos de atributos georreferenciados para permitir consultas en MongoDB (7 horas)	(7 de junio) Extender la funcionalidad de MongoDB en Thingsboard agregando el filtro geoIntersects para realizar su	(8 de junio)	(9 de junio)	(10 de junio) Crear backend para enviar fotos al frontend (3 horas), Crear backend para retornar las foto georeferenci

				verificación (6)			adas por fecha (3 horas)
3	(11 de junio) El usuario debe poder cargar el número que desee de imágenes con su respectivos metadatos para persistirlos en MongoDB (5)	(12 de junio) Crear el backend que permite recibir las imágenes y clasificarla según la finca(5 horas)	(13 de junio)	(14 de junio) Extraer metadatos y verificar si está contenido en algún lote (6 horas)	(15 de junio)	(16 de junio) Conectar front-end con back-end para envío de archivos (imágenes) (5 horas)	(17 de junio)
4	(18 de junio) Pruebas con Spark, utilizando Threads (6 horas)	(19 de julio) Crear modelo de ML sobre fotos de hojas y tallos de fotos para clasificarlos como riesgo de enfermedad (9 horas)	(20 de julio) Investigación del funcionamiento de Apache Spark (6 horas)	(21 de julio)	(22 de julio) Cambio de los diferentes tipos de foto en el mapa según metadatos (7 horas)	(23 de julio)	(24 de julio)
5	(25 de junio) Verificar si hay fotos sobre una fecha en específico (6 horas)	(26 de junio)	(27 de junio)	(28 de junio) Dibujar un polígono sobre una foto (3 horas)	(29 de junio)	(30 de junio) Verificación de datos contenidos en mongoDB para generar etiqueta (5 horas)	(1 de julio) Persistir la foto en mongoDB (5 horas)

6	(2 de julio) Hablar con el estudiante de ingeniería electrónica para conexión de dispositivos desde el invernadero hasta servidores (6 horas)	(3 de julio) Hablar con mis compañeros en cómo repartirnos las tareas para avanzar con el libro de PGR, adelantar cosas que ya teníamos (5 horas)	(4 de julio) Hacer correcciones pedidas del profesor Alexander en la interfaz (7 horas)	(5 de julio)	(6 de julio)	(7 de julio) Arreglar última versión de Thingsboard en el servidor (5 horas)	(8 de julio)
7	(9 de julio) Crear modelo de predicción para problemas en imágenes detalladas sobre hojas (8 horas)	(10 de julio) Realización de la cartelera para la vitrina (4 horas)	(11 de julio) Manual de instalación Thingsboard, y edición del poster (5 horas)	(12 de julio) Crear modelo de análisis de imágenes operando fotos nir / red, terminación del póster para la vitrina (7 horas)	(13 de julio) Obtener recorte de la imagen del polígono hecho por el usuario. Envío de datos desde el invernadero hasta los servidores donde está la plataforma Thingsboard (6 horas)	(14 de julio)	(15 de julio)
8	(16 de julio)	(17 de julio) Realización de vídeo informativo de la herramienta Thingsboard para mostrar en vitrina (4 horas)	(18 de julio) Montaje de vitrina en la escuela (3 horas), realización de diapositivas para presentación de vitrina (2 horas)	(19 de julio) Vitrina académica (6 horas)	(20 de julio)	(21 de julio) Recursos (RESTful API) Landlots (3 horas)	(22 de julio) Unir la parte del cálculo del NDVI en la función de etiqueta (8 horas)

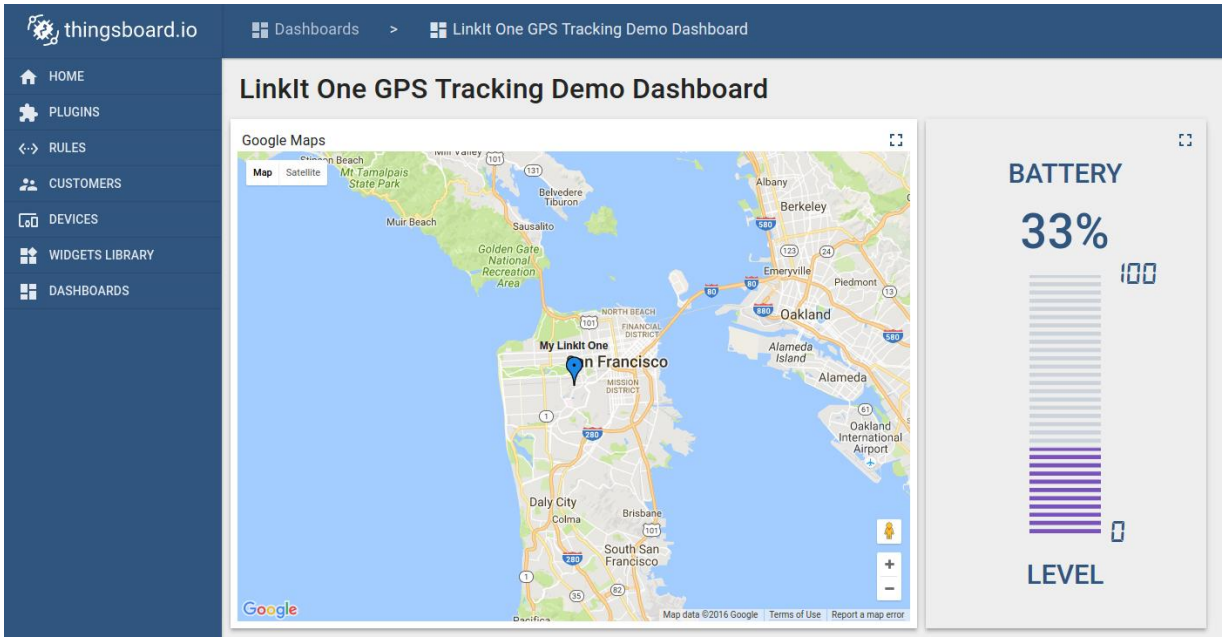
Tabla final

	Fecha	Horas
1	30/05 - 05/06	23
2	06/06 -12/06	26
3	13/06-19/06	26
4	20/06-26/06	19
5	27/06-03/07	24
6	04/07-10/07	24
7	11/07-17/07	22
8	18/07-24/07	19

2. Visión de producto

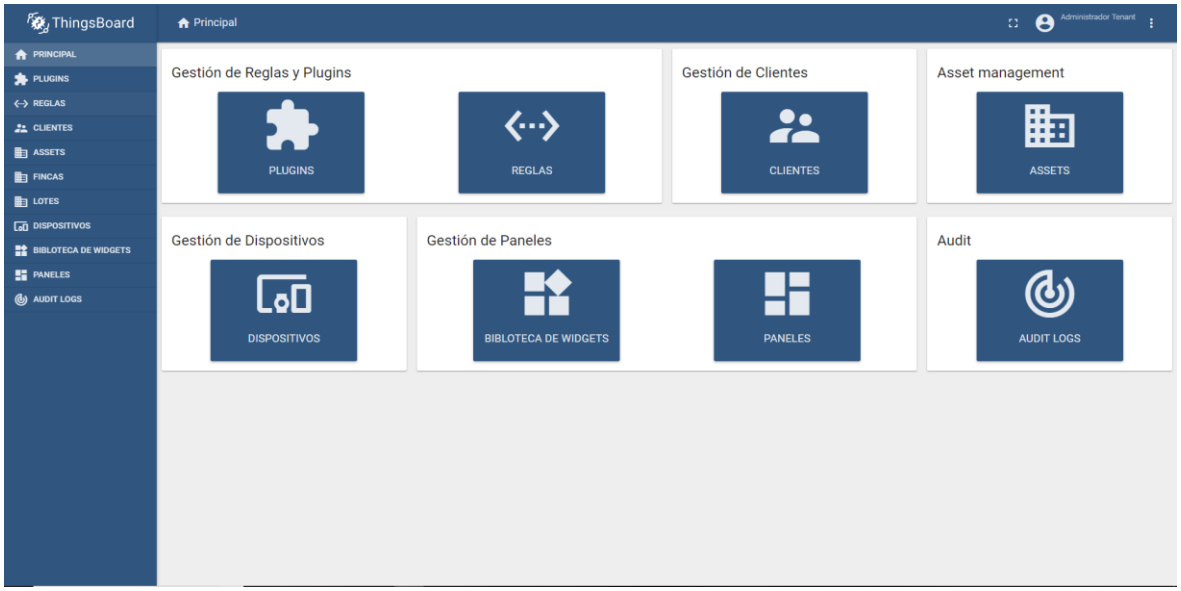
2.1. Descripción de la funcionalidad original de la plataforma Thingsboard

Thingsboard es una plataforma IoT (Internet de las cosas), de código abierto que permite la recolección, procesamiento y visualización de datos y administración de dispositivos. Esto permite la conectividad de dispositivos vía a protocolos estándar IoT tales como: MQTT, HTTP y CoAP, soportando despliegues tanto en la nube como en infraestructura dentro de una empresa.

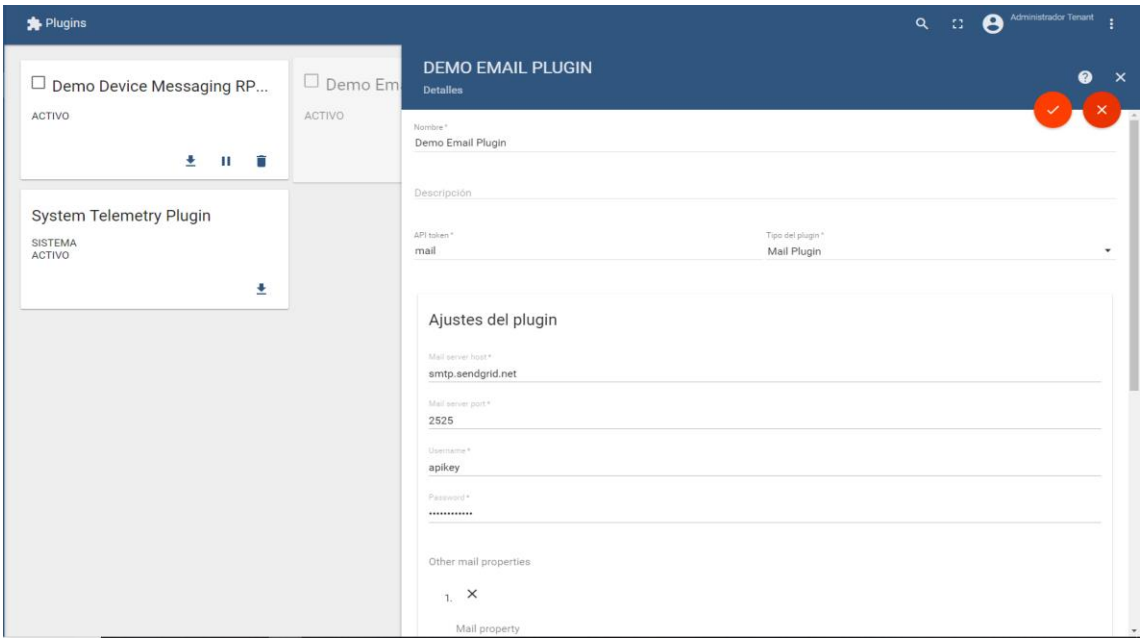


La plataforma Thingsboard cuenta con 8 funcionalidades principales:

1. **Principal**: Muestra el menú con las 8 funcionalidades de Thingsboard que tiene en la barra lateral izquierda, pero en formato más grande.



2. **Plugins**: Los plugins sirven para poder controlar el comportamiento del sistema, estos se encargan de comunicarse con servidores externos a la plataforma para poder recibir datos, realizar acciones con ellos. Estos plugins contienen “reglas” (que son otra funcionalidad principal de Thingsboard) con las cuales manipula los datos.
   Con los plugins se puede:
   - Procesar mensajes de dispositivos.
   - Procesar llamados REST API de aplicaciones de servidores externos
   - Comunicarse con aplicaciones de servidores externos utilizando websockets.
   - Comunicarse entre instancias de un mismo plugin en el clúster de Thingsboard usando llamadas RCP asíncronas.
   - Persistir y consultar eventos, datos de telemetría y atributos de dispositivos.

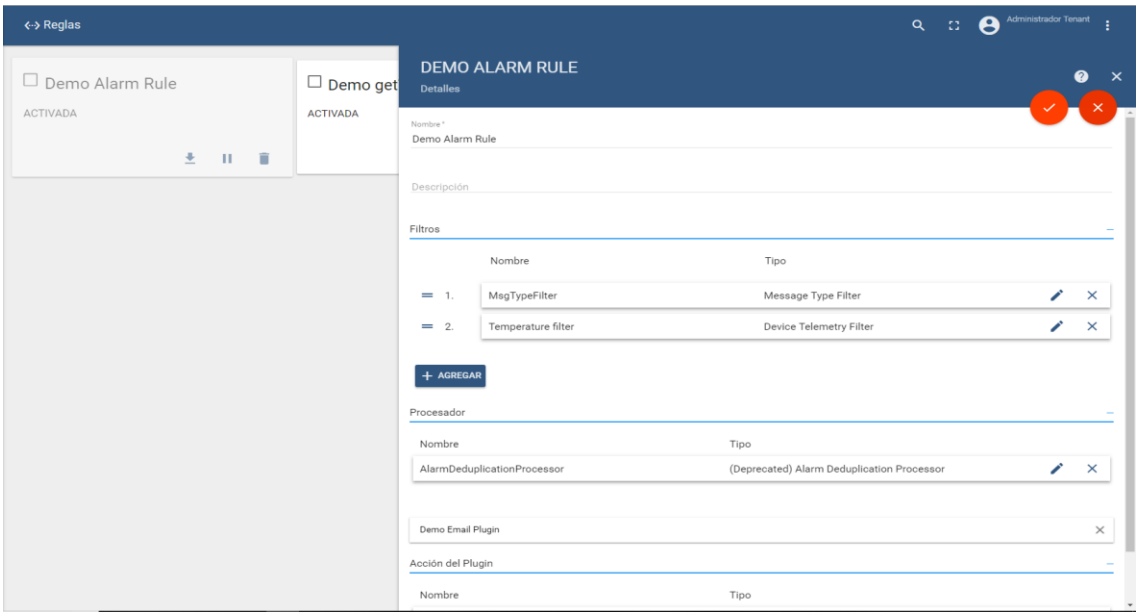


3. **Reglas**: Las reglas consisten en tres componentes principales: los filtros, los procesadores y las acciones. Dependiendo de la implementación, cada componente puede requerir una configuración antes de ser utilizado. Para configurar una regla, se necesita configurar, por lo menos, un filtro y una acción, los procesadores son opcionales.

a. **Filtros**: Como su nombre lo dice, se encargan de filtrar datos de mensajes entrantes, ya que si la información cumple cierta o ciertas condiciones, el filtro permite el paso de esos datos, de lo contrario, los ignora.

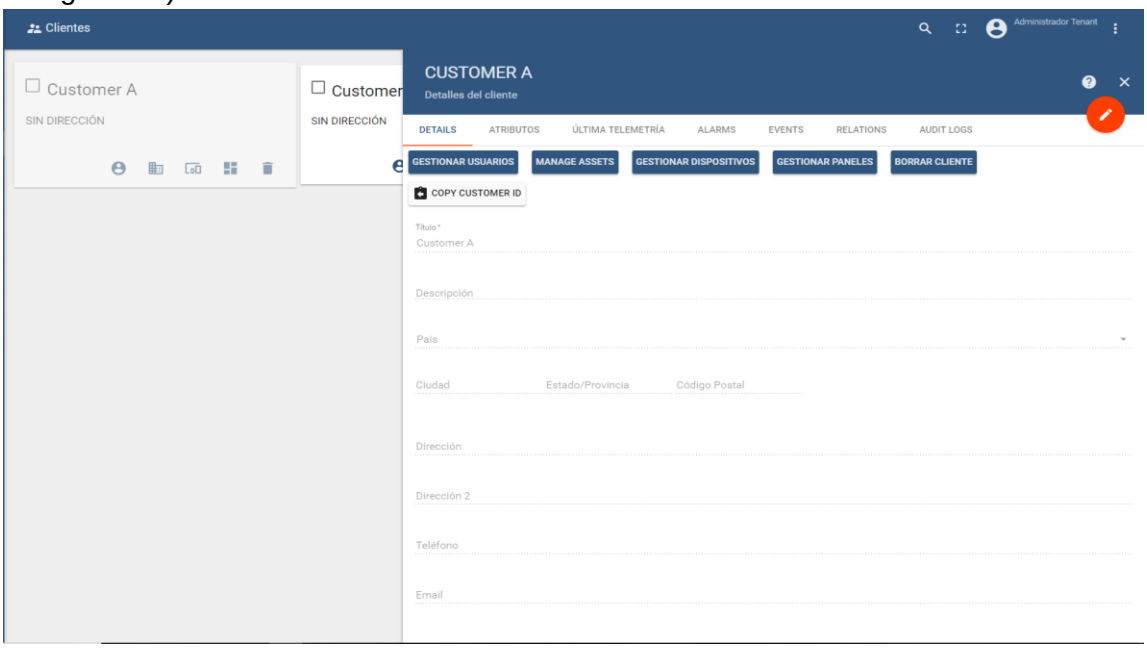
b. **Procesadores**: Se encargan de de procesar y analizar mensajes entrantes y añaden metadatos o etiquetas ellos, dándole más facilidad a la plataforma para identificarlos.

c. **Acciones**: Son los responsables de convertir mensajes entrantes con metadatos a mensajes personalizados que son entregados a ciertos plugins. Las acciones realizan hechos si los mensajes o datos cumplen ciertas condiciones que van desde enviar mensajes por correo hasta la activación de alarmas para que el usuario haga algo.

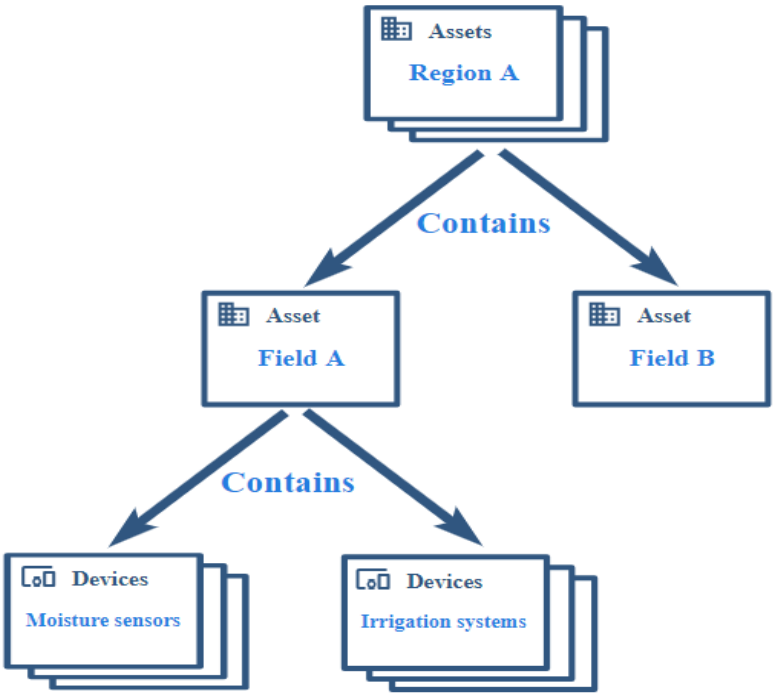


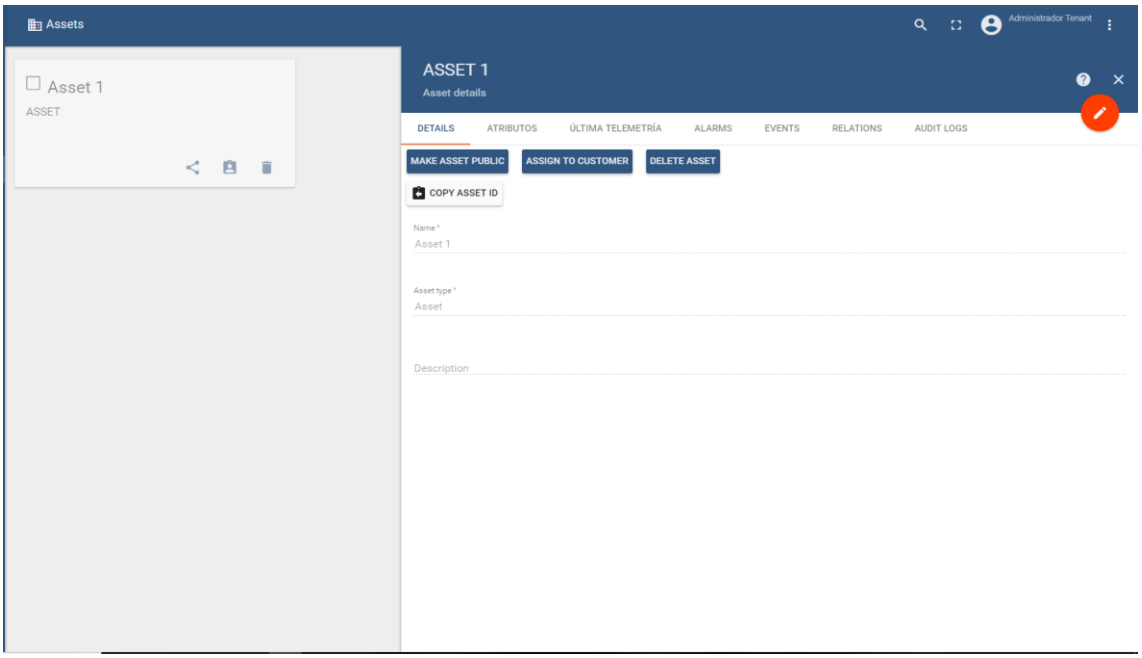
En la página oficial de Thingsboard.io se encuentra la forma de comunicarse con un aplicación Spark externa a la plataforma para que analice los datos (https://thingsboard.io/docs/samples/analytics/spark-integration-with-thingsboard/).

4. **Clientes**: Son objetos que representan a clientes o personas las cuales se les puede identificar colocando toda la información personal del cliente (ej: nombre, correo electrónico, teléfono, etc) y de esta forma a este cliente se le puede hacer propietario de sus propios “assets” (una funcionalidad principal de Thingsboard), de sus propios dispositivos o de sus propios paneles o “dashboard” (una funcionalidad principal de Thingsboard).

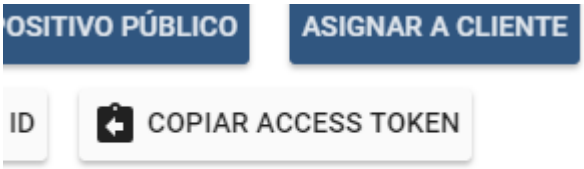


5. **Assets**: Sirven para poder organizar de forma jerárquica los dispositivos de un usuario, ya que un asset puede tener muchos dispositivos o muchos assets, de tal forma que un asset puede representar un terreno que contiene terrenos y que a su vez esos terrenos contienen dispositivos.

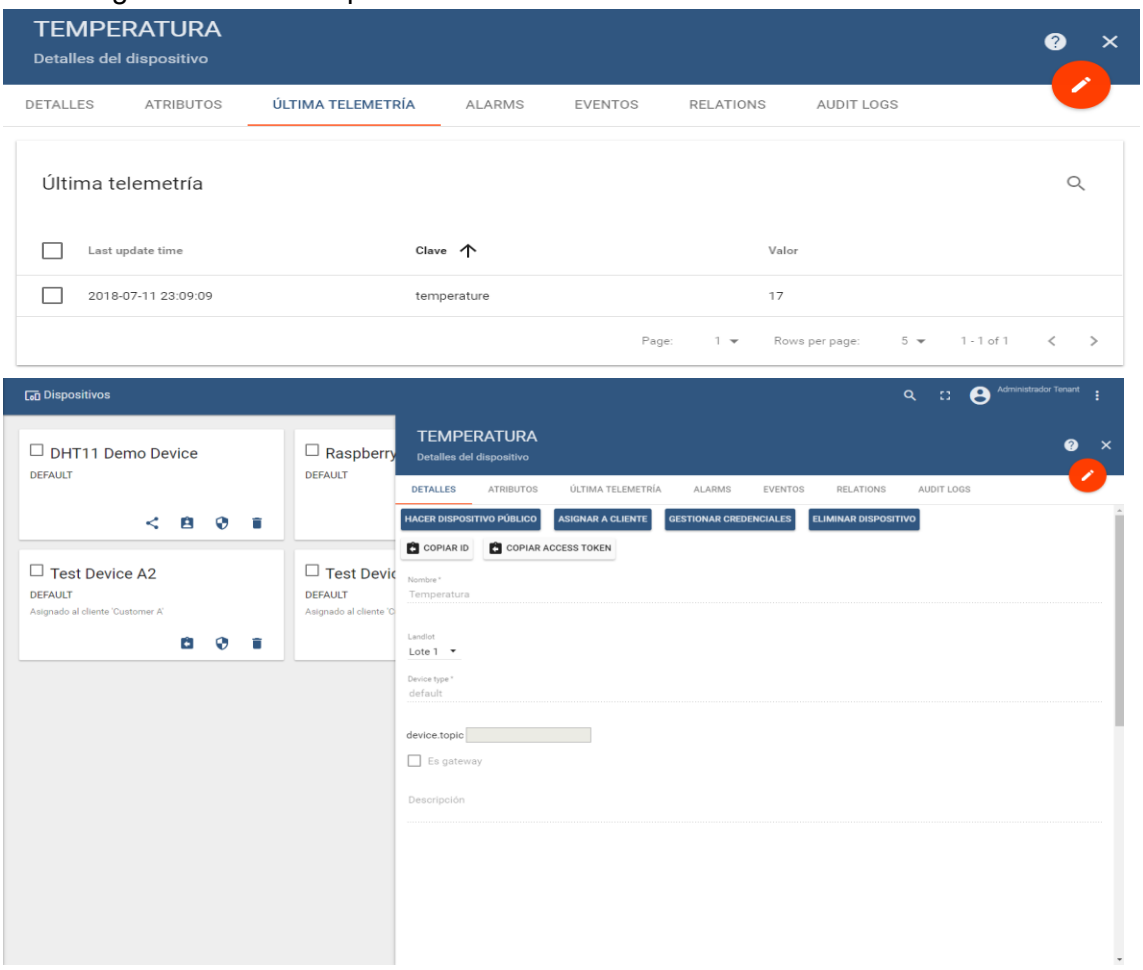




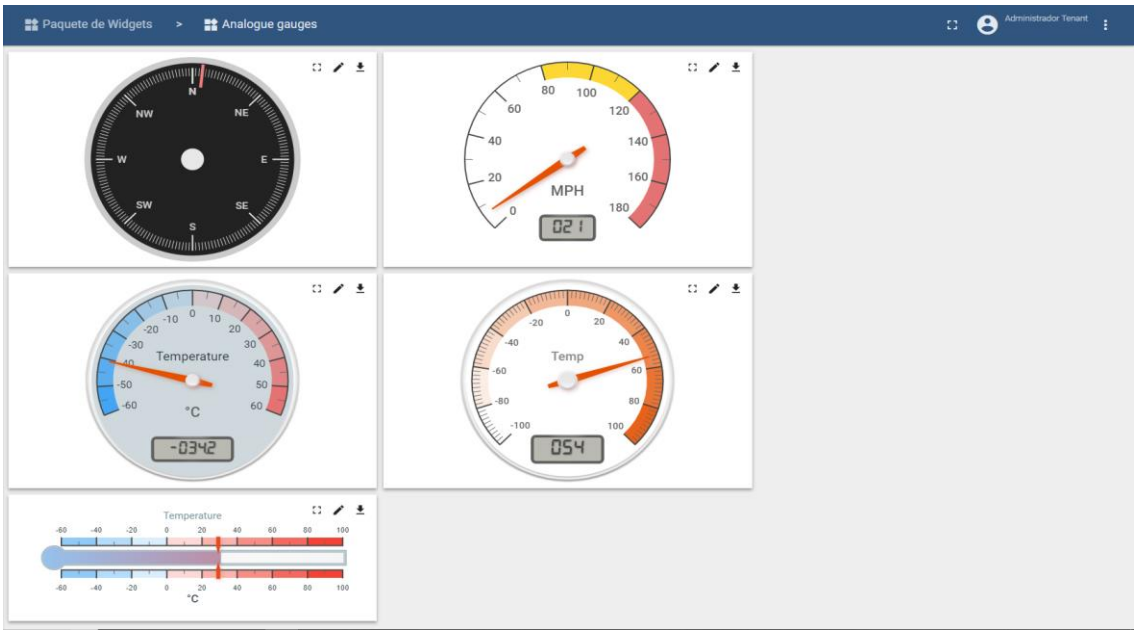
6. **Dispositivos**: Se encargan de recibir los datos de sensores o dispositivos reales, estos dispositivos se comunican a través de los protocolos MQTT, HTTP o CoAP o por el Gateway IoT de Thingsboard. Para poder enviar datos a los dispositivos toca tener un "access token" que el mismo dispositivo provee.



Ya cuando se agregue el access token y se envíen los datos, se podrán observar que están llegando mirando la pestaña de "Última telemetría".



7. **Biblioteca de widgets**: Un widget es un objeto que le permite al usuario poder visualizar datos de telemetría, alarmas, mapas, entre otros objetos de forma mucho más amigable y elegante. Por lo tanto, la biblioteca de widgets sirve para crear, personalizar y guardar los widgets hechos por el usuario, para poderlos usar posteriormente.



8. **Paneles (dashboard)**: Los paneles o dashboard sirven para poder tener mejor control sobre las funcionalidades anteriormente mencionadas, ya que con ayuda de los widgets el usuario puede observar de manera más detallada y amigable los estados de los dispositivos, alarmas, mapas, entre otros, todos reunidos en una misma pantalla o como lo quiera establecer el usuario.

2.2. Descripción de la extensión requerida para la plataforma Thingsboard

Indexación geoespacial Validar la referenciación de terrenos a partir de su ubicación en un API de mapas.	**Modelo de información extendido** Se maneja el concepto de Finca, lotes y cultivos
Manejo de imágenes multiespectrales Almacenamiento e indexación de imágenes a través de GridFS	**Integración con Spark** Analizar datos en tiempo real y generación de alarmas y acciones
Machine Learning Entrenamiento de modelos para hacer predicciones	**Etiquetas espacio temporales** Clasificación de características sobre datos históricos realizadas por un experto

1. Dentro de la implementación llevada a cabo en Thingsboard, se incorporan los conceptos de finca, lote y cultivo permitiendo al administrador del sistema tener un control más detallado de su propiedad, además de recolectar información detallada de la finca como lo es la estructura, datos de cada habitante, climatología local, entre otros.
2. Es necesario solicitar al usuario las coordenadas geoespaciales de cada uno de los conceptos previamente mencionados, para posteriormente registrar los dispositivos que se van a agregar al sistema y que recolectarán los datos de sensores que marcaran sus datos de telemetría en los cultivos implantados, la plataforma válida que los datos geoespaciales pertenezcan a las limitaciones del terreno y no generar inconsistencias a la hora de establecer su configuración, es decir, al momento de registrar un lote se valida que sus coordenadas están contenidas sobre la finca seleccionada como los dispositivos sobre los lotes.
3. La aplicación permite extraer metadatos de imágenes tomadas por drones con el objetivo de conocer el estado de un cultivo a lo largo de un periodo de tiempo; permite al usuario etiquetar una región del cultivo y señalar algún problema que puede afectar el desarrollo del mismo.
4. Una vez se establezcan los sensores reales sobre un lote, se deben crear los dispositivos en Thingsboard, el cual, cada sensor debe enviar sus datos a través de un tópico y el token correspondiente para que se puedan recibir los datos en la plataforma, una vez llegan estos datos se ejecutan las siguientes acciones; 1. Los datos se persisten en la base de datos Cassandra, 2. Se publican en el tópico respectivo de Kafka el cual actúa como una cola de mensajes bajo el patrón de publicador - suscriptor, y por último se mostrarán en pantalla a través del Front-End de Thingsboard.

   Las aplicaciones de spark estará cada n segundos revisando sus respectivos tópicos de kafka (el n varía según el tiempo de transmisión que demoran los sensores en reportar los datos de telemetría), una vez estén toda la información almacenada, se procesaran los datos para separarlos por cultivo, se promedian estos datos y se evalúan las reglas definidas por el usuario, en caso de que alguna

regla de como resultado una probabilidad de enfermedad, se disparan las acciones asociadas a la regla, las reglas pueden ser algoritmos o modelos de machine learning y pueden utilizar info en cassandra (datos históricos) o mongo (datos georreferenciados) según sea necesario.

Para consultar en tiempo real el estado de la finca, se procede a seleccionar el dashboard asociado a la misma para ofrecerle al usuario un control de mando, en donde puede monitorear los cultivos con sensores activos y estar al tanto de alertas que pueden ir activándose si está sucediendo algo fuera de lo normal, cada cultivo tiene un color que determina su estado actual y además de poder implementar reglas que puedan ejecutar acciones tales como el envío de drones para aplicar algún componente sobre el cultivo o un sistema de riego programado.

2.3. Historias de usuario y criterios de aceptación

Historia Usuario	Descripción	Criterios de aceptación
Administrar un cultivo de un lote	Como administrador de los lotes Quiero poder cambiar el estado del cultivo asociado a un lote Para que los indicadores calculados por la plataforma sean consistentes con lo que físicamente se tiene en el lote	Establecer una lista de chequeo de buenas prácticas
Registrar una finca	COMO Administrador de una finca QUIERO registrar una finca PARA PODER realizar el seguimiento de todos los lotes, sensores, cultivos, etc... asociados a la misma y así tomar mejores decisiones a partir de sus resultados.	Poder agregar una foto de la fachada de la finca, para identificarla más fácil
Registrar sensor en un lote	COMO Administrador de la finca QUIERO asociar un sensor a un lote PARA PODER monitorear y registrar los datos para alertar eventualmente sobre un riesgo que pueda estar asociado al cultivo.	Ubicar el dispositivo por medio de un Punto sobre un mapa que debe estar asociado a un lote y contenido en este.
		El usuario puede ver tanto el último como el historial de valores de telemetría ofrecidos por el dispositivo.

Configurar dashboard de una finca	COMO administrador de una finca QUIERO consultar y monitorear el estado actual de los lotes, cultivos, dispositivos, entre otros PARA PODER tomar decisiones.	Al seleccionar el dashboard de una finca se debe mostrar el mapa del terreno que le corresponde
		En el mapa de la finca se debe mostrar los lotes que le pertenecen
		En el mapa de la finca, dentro de los lotes, deben aparecer los dispositivos que están asociados (representados por puntos)
		Cada punto que representa al dispositivo se tornará de un color distinto (verde y rojo) dependiendo del de la alarma
Registrar Etiqueta	Como Experto en agricultura QUIERO crear etiquetas sobre datos históricos PARA PODER alimentar modelos de clasificación que contribuyan a la predicción de enfermedades	Creación de una barra de tiempo sobre un lote en donde se puedan seleccionar un rango de fechas, se avance por día en la misma, y que muestre una señal cuando haya foto
		Creación de un mapa que se ubique cerca a la ubicación del lote, mostrar sobre el mapa todas las fotos ubicadas correctamente
		Permitir al experto crear un polígono de una región sobre el mapa para el posterior etiquetado
		Se habilita un checkbox en con los 5 tipos de imágenes para que cuando el usuario seleccione 1 tipo, se cambien todas las imágenes dentro del mapa correspondientes a ese tipo
		Mostrar gráficas de los valores de telemetría registrados en el rango de fechas
		Permitir al experto escribir la etiqueta y guardarla en la base de datos de MongoDB

Consultar Fotos de cultivos	Como agricultor QUIERO revisar las fotos de mi cultivo PARA PODER conocer el estado del mismo	Crear un api que permite guardar las fotos tomadas por el dron

3. Arquitectura de la solución

3.1. Glosario de conceptos

1. Finca (Farm): Objeto que representa el terreno perteneciente a un agricultor, el cual puede albergar varios lotes, también posee toda la información otorgada por el mismo agricultor, por ejemplo, la tecnología que posee, el tipo de transporte, los trabajadores, las personas que dependen de esta, las construcciones, servicios públicos, entre otras cosas.
2. Lote (Landlot): Objeto que representa una porción del terreno de una finca, por lo que una finca puede tener varios lotes, cada lote posee un cultivo, un historial de cultivos y métodos para ver el estado del cultivo.
3. Dispositivo (Device): Objeto que representa un sensor físico que mide las propiedades del ambiente y del suelo de un terreno, en este caso un lote, por lo que un lote puede tener varios dispositivos, este objeto es el encargado de recibir los datos tomados por los sensores físicos y propagarlos por la plataforma para posteriormente ser analizados.
4. Cultivo (Crop): Un cultivo representa todas las plantas que se sembraron sobre un terreno, esta posee información como el nombre, la razón de porqué se hizo, el tipo de suelo, el registro de las acciones que se le aplicaron, la fecha de inicio y terminación, una lista de chequeo de las buenas prácticas que se ejercieron sobre el cultivo, entre otras cosas.
5. Tablero de control (Dashboard): Con este objeto se puede ver en tiempo real las condiciones en la que se encuentra la finca, sus lotes y los dispositivos a través de un mapa.
6. Imagen (Image): Objeto que representa a las fotos tomada por el dron sobre un lote específico, cada foto equivale a un grupo de 5 fotos en total, por ejemplo, RGB, infrarrojo, rojo, verde, etc. Y con estas fotos poder hacer análisis del terreno.
7. Machine learning: Algoritmos o modelos los cuales son entrenados a partir de un conjunto de datos hecho de forma supervisada o no supervisada, para que posteriormente realice predicciones sobre sucesos ingresando como parámetros características o datos.

3.2. Modelos de datos

3.2.1. Modelo Cassandra

Todas las variables de las tablas de Cassandra se guardan en formato JSON, son de tipo String a excepción de los identificadores o ID que son de tipo UUID.

1. **Farm** (columnas de la tabla):
    a) Id: Identificador único de cada finca
    b) Tenant_id: Identificador único del usuario tenant quien es el que tiene todos los permisos sobre la herramienta
    c) Customer_id: Identificador único del usuario customer quien es el que tiene ciertos permisos sobre la herramienta
    d) Type: Forma de clasificar el objeto de tipo “farm” (finca).
    e) Additional_info: Información adicional que quiera el usuario agregarle a la finca
    f) Dashboard_id: Identificador único de un dashboard o tablero de mando, cada finca tiene un dashboard donde muestra su estado a través de un mapa.
    g) Farm_details: Detalles propios de una finca donde se almacenan su destinación, sus detalles de uso, una lista de servicios públicos, el transporte para producción y una lista con los diferentes puntos de toma de agua de una finca.
    h) Farm_enviroment: Información que detalla el ambiente de la finca, este objeto almacena información como: la climatología local (temperatura, humedad, lluvia e irradiación solar), la orografía del lugar, la distancia de la finca con su respectiva cabecera municipal, la forma de acceso a la finca y el estado de la carretera.
    i) Home_details: Información que detalla características del hogar de la finca, tales como, el material de construcción de la casa, el material del piso, cuántos cuartos hay, la descripción del baño, la descripción de la cocina, la cantidad de personas que dependen de la finca, la cantidad de trabajadores y una lista con toda la información personal de los habitantes de la casa.
    j) Irrigation_systems: Una lista con toda la información de los sistemas de riego que contiene una finca.
    k) Location_description: Texto que detalla la descripción del lugar o finca
    l) Name: Nombre de la finca.
    m) Seach_text: identificador del nombre de la finca.
    n) Total_area: Detalla la extensión de la finca y este puede estar en hectáreas o fanegadas.
2. **Landlot** (columnas de la tabla):
    a) Id: Identificador único de cada lote
    b) Tenant_id: Identificador único del usuario tenant quien es el que tiene todos los permisos sobre la herramienta
    c) Customer_id: Identificador único del usuario customer quien es el que tiene ciertos permisos sobre la herramienta
    d) Type: Forma de clasificar el objeto de tipo “landlot” (lote).
    e) Additional_info: Información adicional que quiera el usuario agregarle al lote
    f) Crop: Cada lote tiene su propio cultivo y en este se guarda la información tal como: el nombre del cultivo, el porqué del cultivo, causa del cultivo, fecha de inicio del cultivo, fecha de terminación del cultivo, número de semanas que lleva el cultivo, condiciones iniciales del terreno, una lista de acciones que se le han hecho al cultivo,

sabe si se ha terminado o no el cultivo, estado del cultivo y una lista con las buenas prácticas que se aplicaron sobre ese cultivo.

g) Crop_history: Lista con todos los cultivos terminados que ha tenido el lote.
h) Devices: lista con los dispositivos o sensores que se encuentran en ese terreno
i) Farm_id: identificador único de una finca, cada lote conoce a su finca, por lo que una finca contiene muchos lotes.
j) Ground_features: Características del suelo, que detallan datos como: la densidad del suelo, la compactación del suelo, la inclinación y los datos hidrológicos.
k) Name: Nombre del lote.
l) Search_text: Identificador del nombre del lote.
m) Total_area: Detalla la extensión del lote y este puede estar en hectáreas o fanegadas.

3. Device (columnas de la tabla):

a) Id: Identificador único para cada dispositivo.
b) Tenant_id: Identificador único del usuario tenant quien es el que tiene todos los permisos sobre la herramienta
c) Customer_id: Identificador único del usuario customer quien es el que tiene ciertos permisos sobre la herramienta
d) Type: Forma de clasificar el objeto pero en este caso pueden haber dos tipos de dispositivo, uno de tipo “Spark”, quien es el que va a analizar los datos recogidos por los dispositivos de tipo “default”
e) Additional_info: Información adicional que quiera el usuario agregarle al dispositivo.
f) Landlot_id: Identificador único de cada lote, un dispositivo conoce a su lote, por lo que un lote puede tener muchos dispositivos.
g) Name: Nombre del dispositivo.
h) Search_text: Identificador del nombre del dispositivo.

4. Dashboard (columnas de la tabla):

a) Id: Identificador único de cada dashboard.
b) Tenant_id: Identificador único del usuario tenant quien es el que tiene todos los permisos sobre la herramienta
c) Assigned_customers: Customers o clientes que pueden ver el dashboard.
d) Configuration: Objeto del dashboard donde detalla la información de sus widgets, los cuales son objetos que permiten ver la información de los sensores, alarmas, mapas, entre otros de manera más amigable para el usuario
e) Search_text: Identificador del nombre del dashboard.
f) Title: Titulo o nombre del dashboard.

5. ts_kf_cv (columnas de la tabla) (esta tabla es la que guarda la información de los datos enviados por los sensores):

a) Entity_type: Tipo del dispositivo que envía la información
b) entity_id: Identificador único del dispositivo que envía la información

c) Key: Tipo de dato que envía el dispositivo (ej: temperature, air_humidity, wind_speed).
d) partition : Fecha que hace referencia al primer día del mes, cuando el dispositivo envió el dato.
e) Ts: Fecha en el que el dispositivo envía el dato.
f) Bool_v: Es el dato que almacena la tabla si el dispositivo envía valores booleanos.
g) Dbl_v: Es el dato que almacena la tabla si el dispositivo envía valores double.
h) Long_v: Es el dato que almacena la tabla si el dispositivo envía valores long.
i) Str_v: Es el dato que almacena la tabla si el dispositivo envía valores en texto.

6. Plugin (columnas de la tabla):
a) Id: Identificador único de cada plugin
b) Tenant_id: Identificador único del usuario tenant quien es el que tiene todos los permisos sobre la herramienta
c) Additional_info: Información adicional que quiera el usuario agregarle al plugin.
d) Api_token: Token que representa al plugin
e) Configuration: Información que detalla como el plugin está configurado para establecer conexión o comunicación con otro servidor.
f) Name: Nombre del plugin
g) Plugin_class: Nombre del paquete en donde se está ejecutando un plugin en específico con los estándares de Java (ej: org.thingsboard.server.extensions.core.plugin.rpc.RpcPlugin)
h) Public_access: Se define con un valor booleano si cualquiera puede o no acceder al plugin.
i) Search_text: Identificador del nombre del plugin.
j) State: Palabra que especifica el estado del plugin, si está activo o no.

7. Rule (columnas de la tabla):
a) Id: Identificador único de cada regla
b) Tenant_id: Identificador único del usuario tenant quien es el que tiene todos los permisos sobre la herramienta
c) Action: Lista de acciones que debe realizar la regla si se cumplen ciertas condiciones.
d) Additional_info: Información adicional que quiera el usuario agregarle al dispositivo.
e) Filters: Filtros que le permiten a la regla poder escoger la información que considera importante de los datos que le envían.
f) Name: Nombre de la regla.
g) Plugin_token: Token que representa a un plugin, por lo que una regla conoce a un plugin.
h) Processor: Lista de procesadores que tiene una regla, los cuales se encargan de analizar la información que reciben.
i) Search_text: Identificador del nombre de la regla.
j) State: Estado de la regla para decir que está activa o no.

8. Alarms (columnas de la tabla):

a) Tenant_id: Identificador único del usuario tenant quien es el que tiene todos los permisos sobre la herramienta
b) Originator_id: Identificador del dispositivo o asset que causó la alarma.
c) Originator_type: Tipo del objeto que causó la alarma (device o asset)
d) Type: Tipo de la alarma
e) Id: Identificador único de cada alarma.
f) Ack_ts: Fecha en que la alarma ya fue vista
g) Clear_ts: Fecha en que la alarma fue borrada
h) Details: Detalles o información que le da el usuario a la alarma
i) End_ts: Fecha de terminación de la alarma
j) Propagate: Un valor booleano que indica si la alarma se propagó por la herramienta
k) Severity: Nivel de severidad de la alarma que puede ir desde leve hasta una crítica
l) Start_ts: Fecha de inicio de la alarma
m) Status: Estado de la alarma

3.2.2. Modelo MongoDB

El rol que desempeña MongoDB en la extensión de la plataforma de Thingsboard abarca la indexación geoespacial, el cual permite almacenar y consultar datos geográficos de fincas, lotes y dispositivos así como validar que la información suministrada esté correcta tales como Lotes contenidos en Fincas o Dispositivos en Lotes, para consultar los otros tipos de objetos GeoJSON que permite utilizar los modelos de MongoDB, también se utiliza GridFS para el almacenamiento y recuperación de archivos que exceden el tamaño limite de 16MB de documentos BSON, es decir que nos permite almacenar archivos de todo tipo dividiéndolo por partes o “chunks”, cada porción tendrá un identificador y consecutivo que permitirá reconstruir el archivo en el momento que se desee recuperar, así mismo está la opción de agregar metadatos para incluir información sumamente importante para un objetivo en concreto.

El concepto de ObjectId que usa MongoDB para asignar identificadores a documentos donde no es especificado, representa un tipo BSON de 12-byte único y el cual se genera basándose en el tiempo, identificador de la máquina, identificador del proceso y contador incremental de procesos locales.

Clases

FARM

Nombre	Tipo	Descripción
_Id	String	Pertenece al mismo identificador que Thingsboard asigna al momento de crear una Finca.
farm_Name	String	Corresponde al nombre que se asignó al crear la finca.
polygon	Polygon	La clase “polygon” está compuesto por el atributo coordinates el cual consiste del conjunto de puntos que encierran una región sobre el mapa y representa la finca que se desea agregar, el otro atributo es el “type” y es el tipo de figura geométrica que corresponde el objeto, en este caso es de tipo “Polygon”.

LANDLOT

Nombre	Tipo	Descripción
_Id	String	Pertenece al mismo identificador que Thingsboard asigna al momento de crear un lote.
landlot_Farm_FK	String	Corresponde al identificador asociado con la finca que encierra el lote creado.
polygon	Polygon	La clase “point” está compuesto por el atributo coordinates el cual consiste de que encierran una región sobre el mapa y representa el lote que debe estar contenido sobre la finca, el otro atributo es el “type” y es el tipo de figura geométrica que corresponde el objeto, en este caso es de tipo “Polygon”.

DEVICE

Nombre	Tipo	Descripción
_Id	String	Pertenece al mismo identificador que Thingsboard asigna al momento de crear un dispositivo.
device_Landlot_FK	String	Corresponde al identificador asociado al lote en donde se establecerá para recibir la información de los sensores.
point	Point	La clase “point” está compuesto por el atributo coordinates el cual consiste de la longitud y latitud que establece un punto sobre el mapa y representa el dispositivo en el que estarán los sensores enviando datos a la plataforma, el otro atributo es el “type” y es el tipo de figura geométrica que corresponde el objeto, en este caso es de tipo “Point”.

SPARK DEVICE

Nombre	Tipo	Descripción
_Id	String	Pertenece al mismo identificador que Thingsboard asigna al momento de crear un dispositivo de tipo Spark.
idLandlot	String	Corresponde al identificador asociado al lote en donde se establecerá para recibir la información de los sensores.
topic	String	El tópico se utiliza para identificar desde la aplicación de spark a cual device de tipo spark enviarle el mensaje de alerta.
token	String	Identificador del dispositivo que al ser creado debe utilizarse para recibir los valores de telemetría enviados por los sensores reales.

FS.FILES

Nombre	Tipo	Descripción
_Id	String	Identificador único al documento creado que contiene toda la información del archivo (foto).
filename	String	Nombre del archivo (foto) que el usuario almacena.
length	Long	Tamaño del documento en bytes.
chunkSize	Long	Representa el tamaño de cada chunk en bytes, GridFS divide el documento en chunks de tamaño “chunkSize”, excepto el último chunk, el cual será tan grande como sea necesario, el tamaño de cada chunk por defecto es de 255 kilobytes (kB).
uploadDate	Date	Fecha del documento cuando se guardó.
md5	String	Representa un hash MD5 del archivo retornado por el comando filemd5 (Se usa este comando para verificar que los archivos están correctamente escritos en MongoDB).
metadata	JSON	Los metadatos son la información adicional que tendrá cada archivo en MongoDB y el usuario puede agregar tantos datos desee, para el desarrollo en Things board, se agregan los siguientes datos: ● **landlotId:** Lote donde se encuentra localizada la foto por sus coordenadas geoespaciales. ● **date:** Fecha en la que fué tomada la foto. ● **coordinates:** Al tomar la foto, se extraen los metadatos asignados por el dron y los cuales posee las coordenadas por grado/minuto/segundo y se realiza su conversión a longitud y latitud.

FS.CHUNK

Nombre	Tipo	Descripción
_Id	String	Identificador único del chunk.
files_id	String	Identificador del documento padre asociado (documento fs.file correspondiente).
n	long	Número de secuencia del chunk.
data	BSON Binary	Información del fragmento correspondiente al archivo según la partición.

3.2.3. Modelo REDIS

Redis es utilizado por las aplicaciones de Apache Spark para compartir datos entre sí, también para almacenar datos que son esenciales a la hora de verificar reglas que tienen en cuenta el factor tiempo, el almacenamiento está basado en tablas Clave/valor, en este caso las claves y valores utilizadas son:

Clave	Valor
humidity+idLandlot	último dato de humedad registrado en el cultivo sobre el lote con id: idLandlot
analysisString+idLandlot	cadena compuesta por '+' ó '-' que representan los momentos en los que se cumple la condición sobre los datos de telemetría en la regla que identifica el tizón tardío
light+idLandlot	último dato de intensidad de la luz registrado en el cultivo sobre el lote con id: idLandlot
start_time+idLandlot	fecha en formato timestamp que identifica el día 0 en que se empezaron a recibir los datos de telemetría al cultivo sobre el lote con id: idLandlot
Warning+idLandlot	valor que indica si hay un periodo de smith en el último día en cultivo sobre el lote: idLandlot
analysisValue+idLandlot	Representación en números de la cantidad de '+' en el analisisString del lote: idLandlot

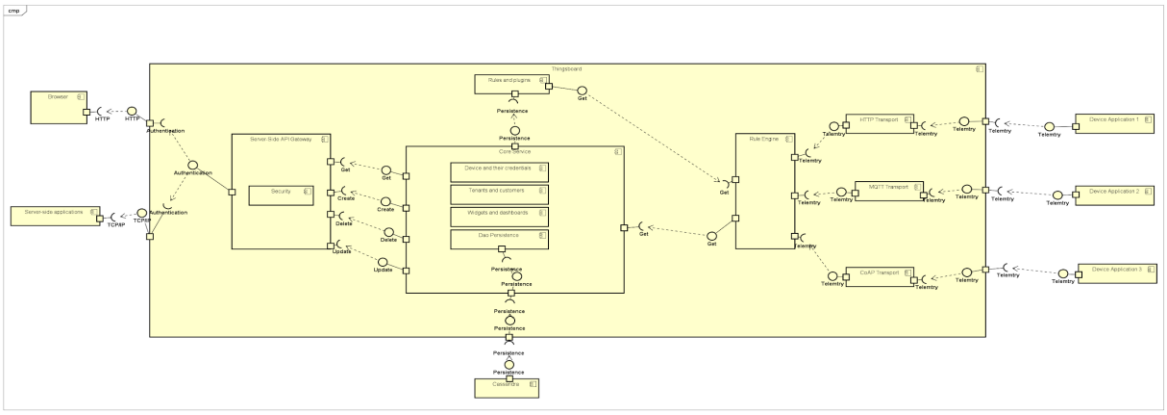
3.3. Recursos (RESTful API/WebSockets API)

RAML anexo en el siguiente link o en el documento anexo llamado (thingsboard.raml):
https://drive.google.com/file/d/1Ty-iyi5OZ_leM6gyWZ93ik0t6jJmRtdQ/view?usp=sharing

3.4. Vistas arquitectónicas

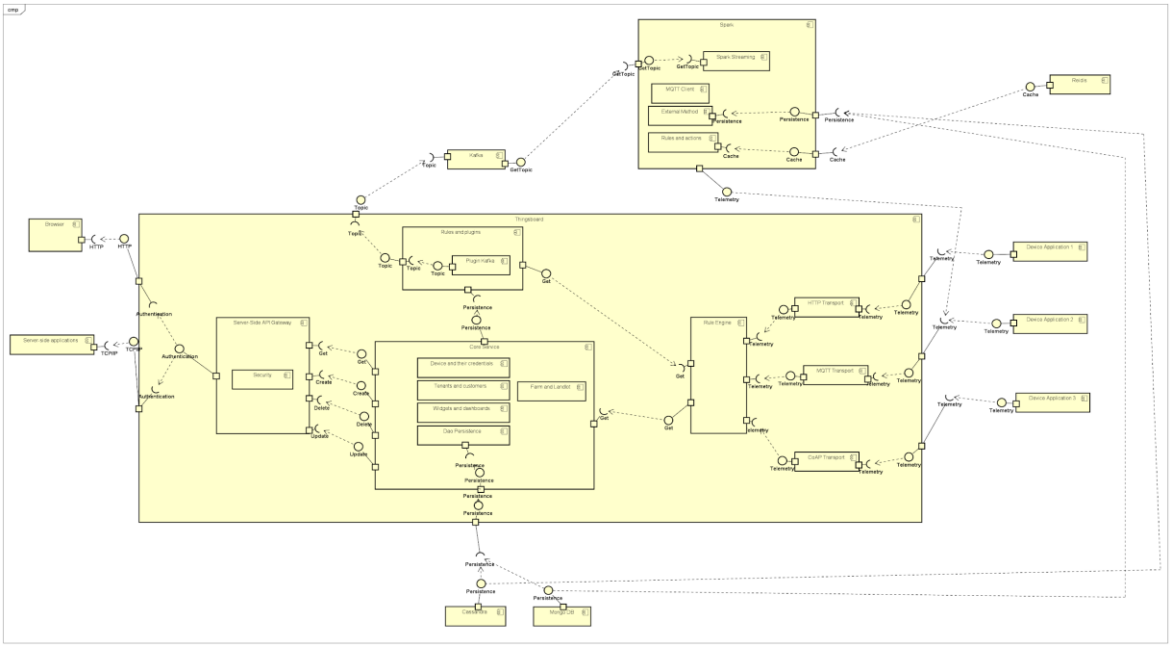
3.4.1. Vista lógica

3.4.1.1 Diagrama de componentes Thingsboard antes de extensión



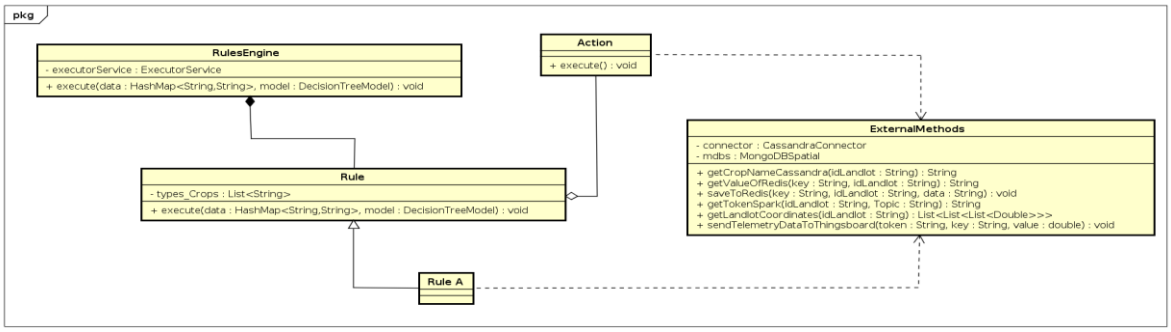
https://drive.google.com/open?id=1FSP74FUOM9OzYAkAzqu9Ns0-CBplmyxl

3.4.1.2 Diagrama de componentes Thingsboard después de extensión



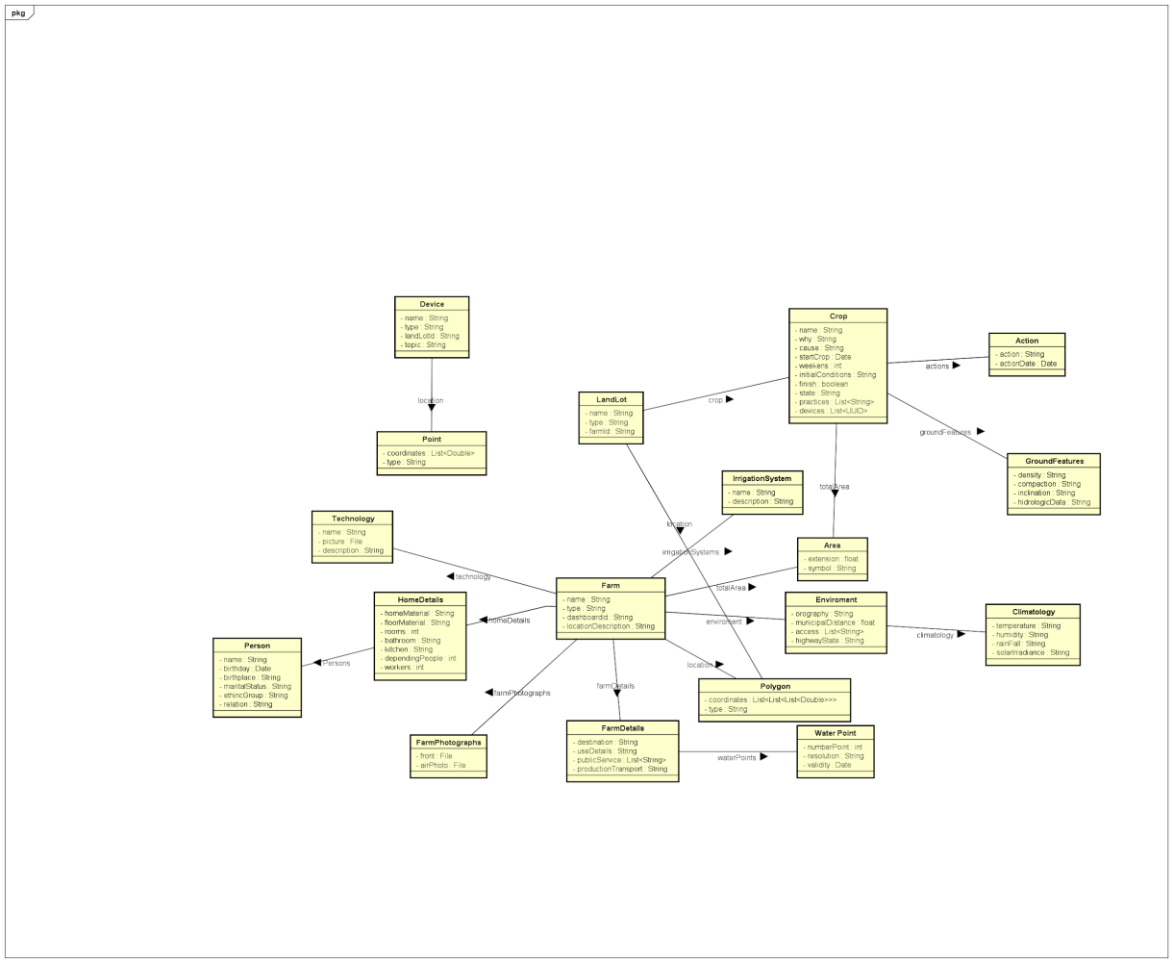
https://drive.google.com/open?id=1e2FkgbcKkW3zQuvNk0tRabyiHFaWIGau

3.4.1.3 Modelo de evaluación de reglas



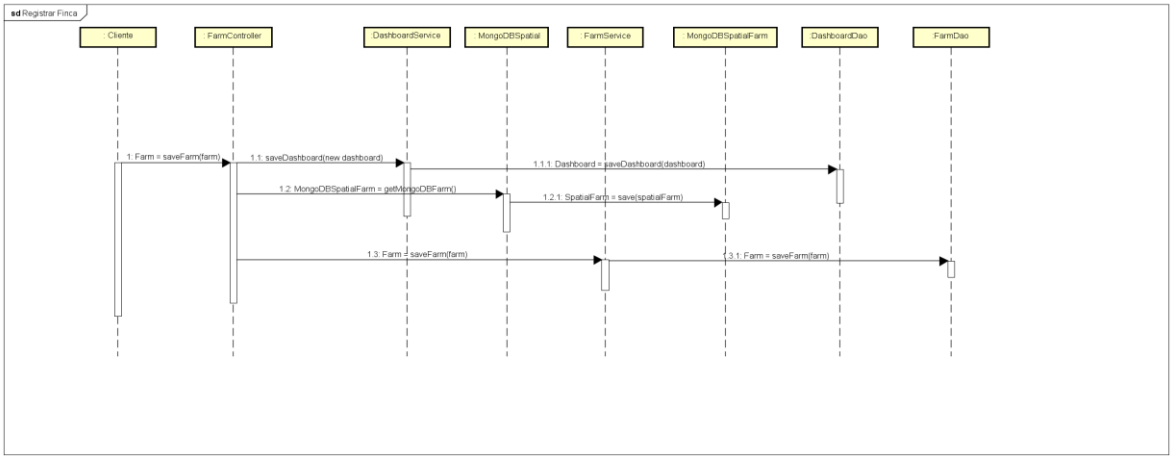
https://drive.google.com/open?id=195TzT-A1Bau0iWFFnxjHVWUTsa5ZYeb1

3.4.1.4 Diagrama de clases Thingsboard



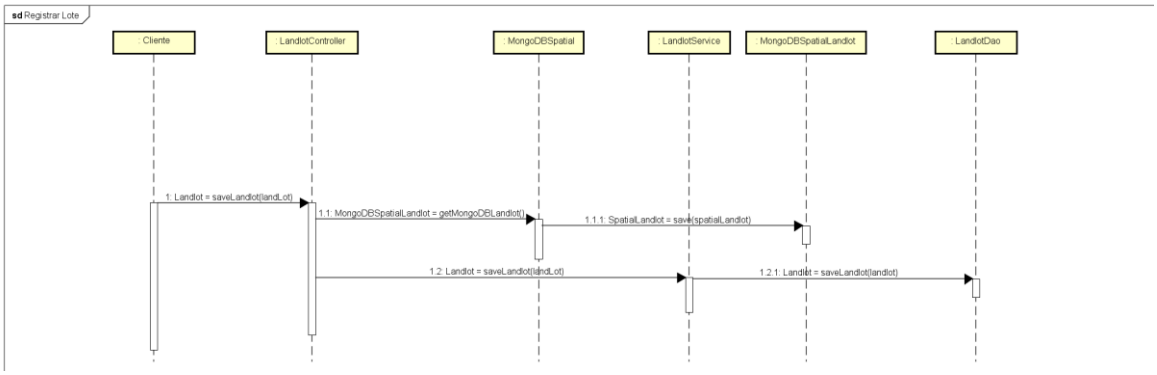
https://drive.google.com/open?id=11H4nD6V2XwFMAe5bFWOwtVLxL1Ya-hoG

3.4.1.5 Diagrama de secuencia registrar una finca



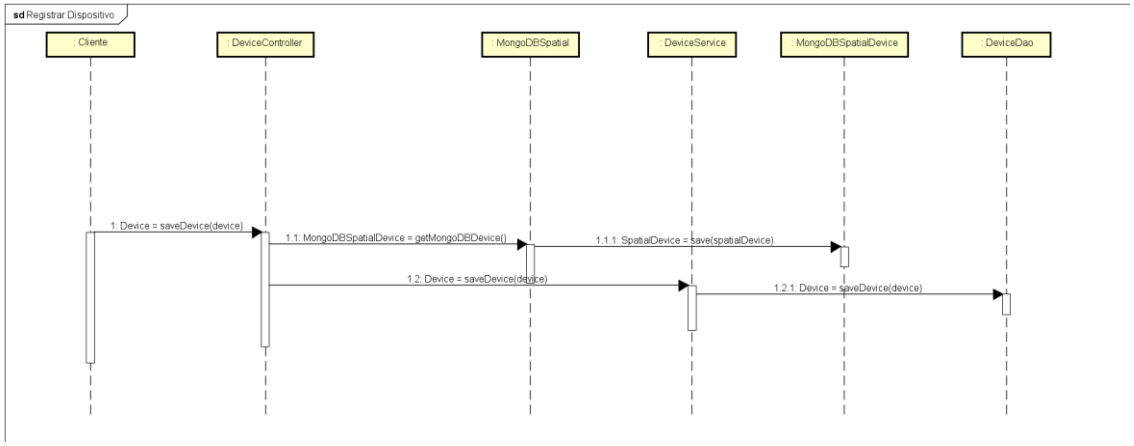
https://drive.google.com/open?id=1eWz3YjP5A5uX95qYeb3ipMTEF5QFmthf

3.4.1.6 Diagrama de secuencia registrar un lote



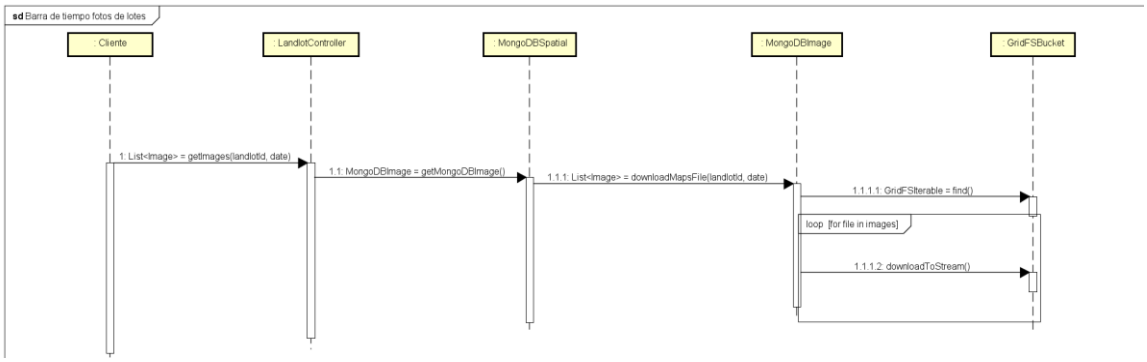
https://drive.google.com/open?id=1tYFt1Yc9iiZpeFpjePau1zMIdqjO2yUS

3.4.1.7 Diagrama de secuencia registrar un dispositivo



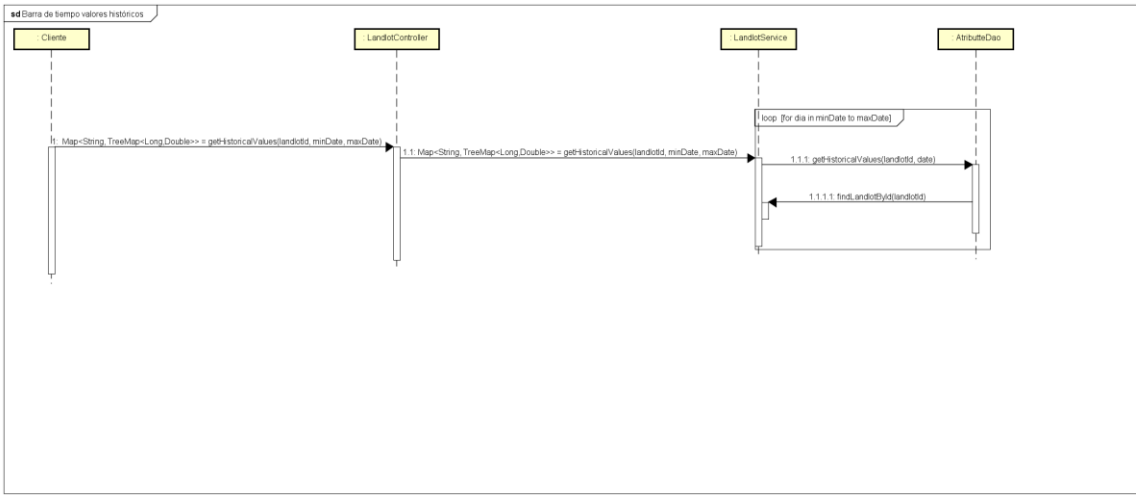
https://drive.google.com/open?id=129FJrFEug1AizWr2XGbgNLIVDEnvJVIE

3.4.1.8 Diagrama de secuencia barra de tiempo foto de cultivos



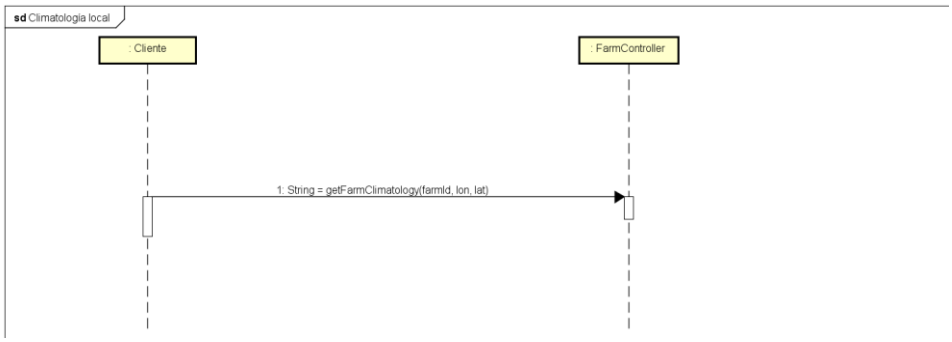
https://drive.google.com/open?id=1ZfYBHO5wuzGZ0bDUlEXDJzazkp75Mq8v

3.4.1.9 Diagrama de secuencia barra de tiempo valores históricos de sensores



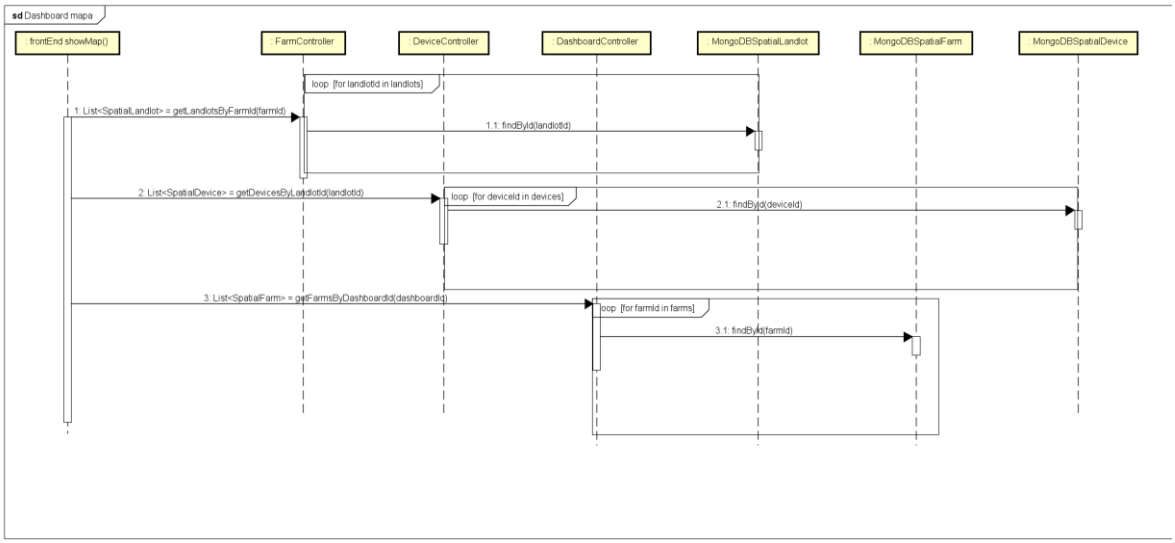
https://drive.google.com/open?id=1jMdTme6hkxNMa9tegtuUMp2g1nSjVjOE

3.4.1.10 Diagrama de secuencia climatología local



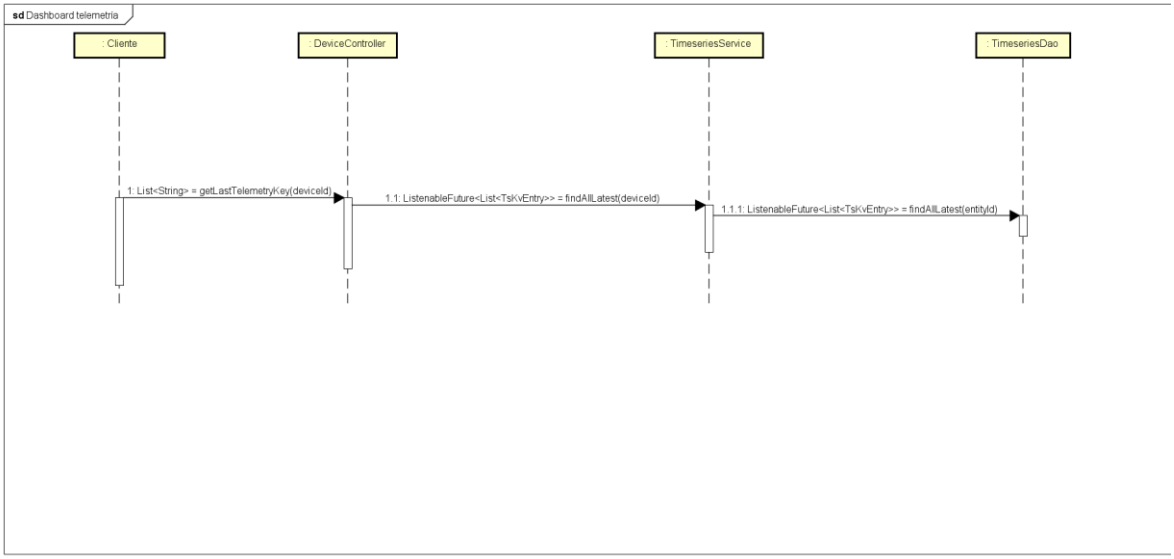
https://drive.google.com/open?id=1wZEH8aRjuPIA10UkRKtQfL7eoaiOCAq1

3.4.1.11 Diagrama de secuencia mostrar mapa en dashboard



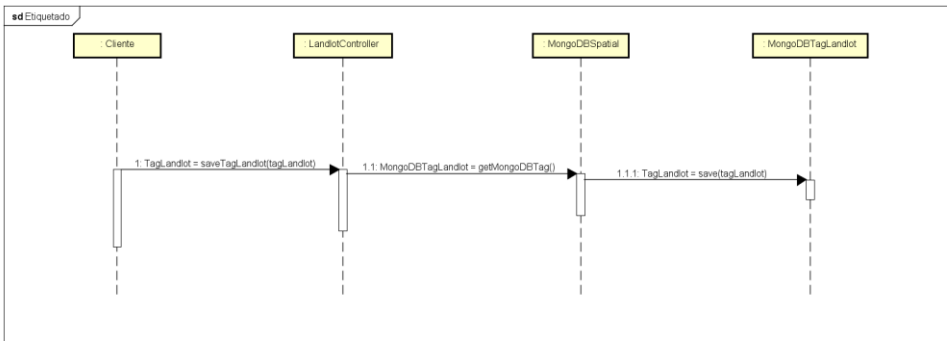
https://drive.google.com/open?id=1OHbUyfRReXVNCwbusMi4oy3qQQXMrL-K

3.4.1.12 Diagrama de secuencia mostrar última telemetría en dashboard



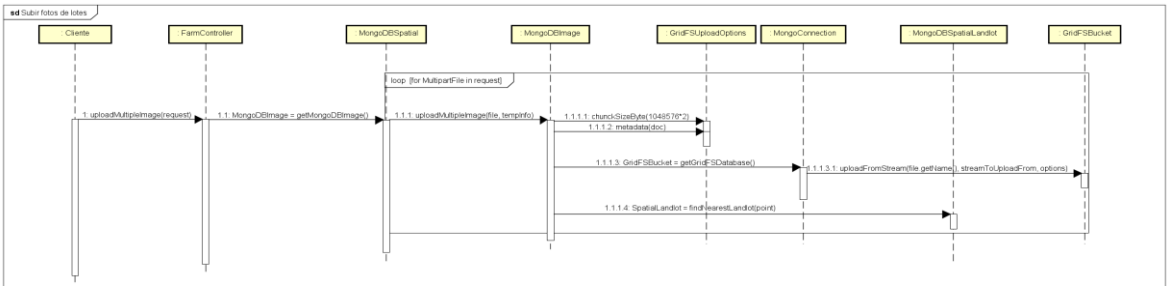
https://drive.google.com/open?id=1NdLYWEYpbfgNYwm4LzXX5Yoi7DnAJgaO

3.4.1.13 Diagrama de secuencia etiquetas espacio-temporales



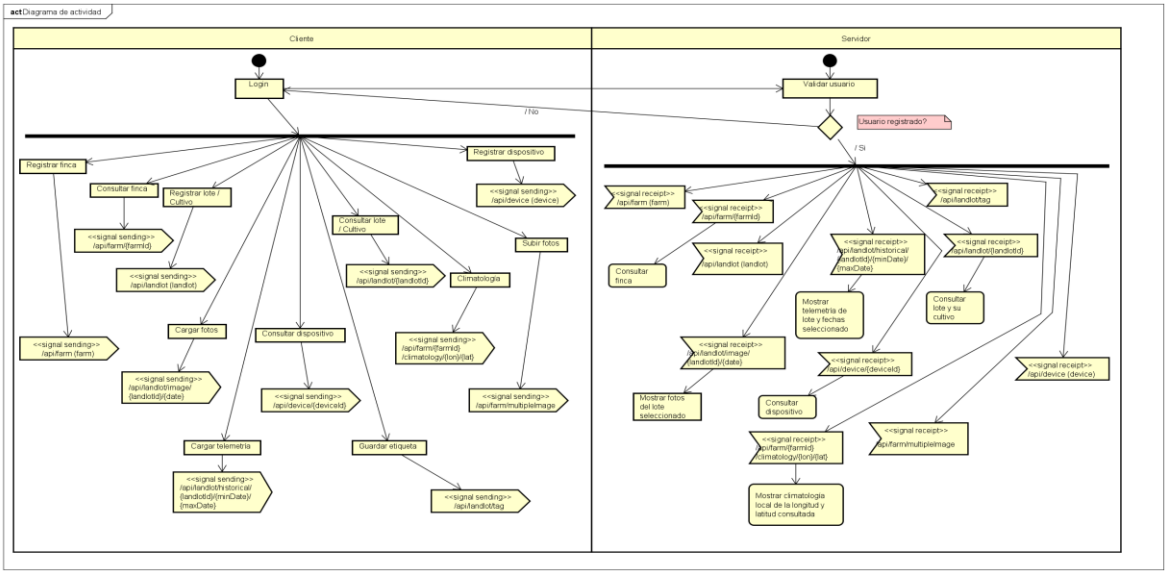
https://drive.google.com/open?id=1Jy7B3BQZTdOXzmcDXmkTRZK2q8Z1R6wE

3.4.1.14 Diagrama de secuencia subir fotos del dron a la plataforma



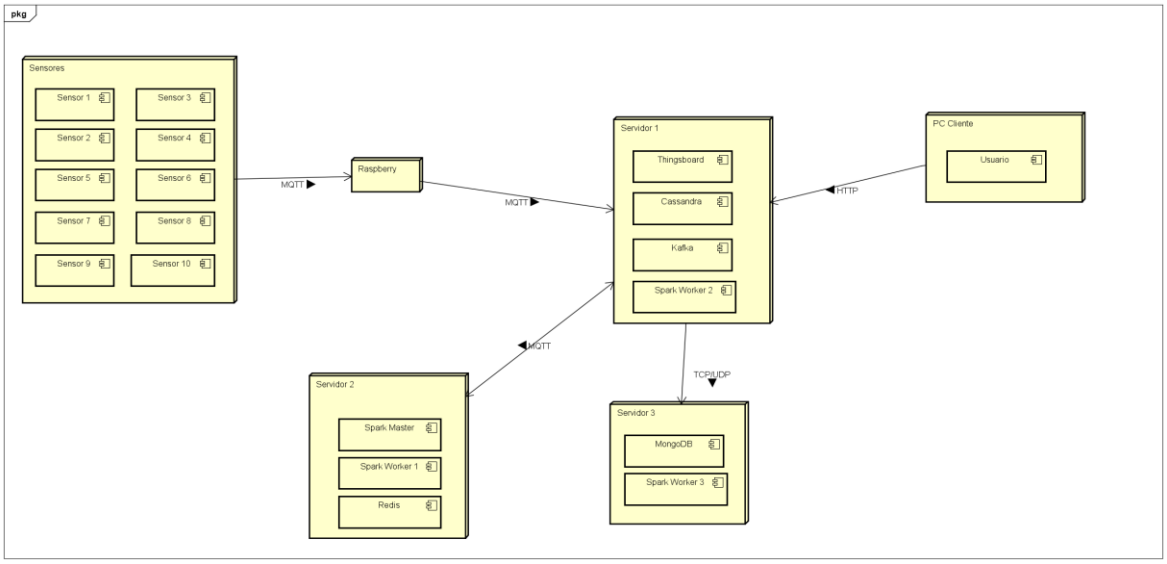
https://drive.google.com/open?id=1gVOqaYS3SGK_-VstTsT8pgF79GN1yBhP

3.4.1.15 Diagrama de actividades de Thingsboard



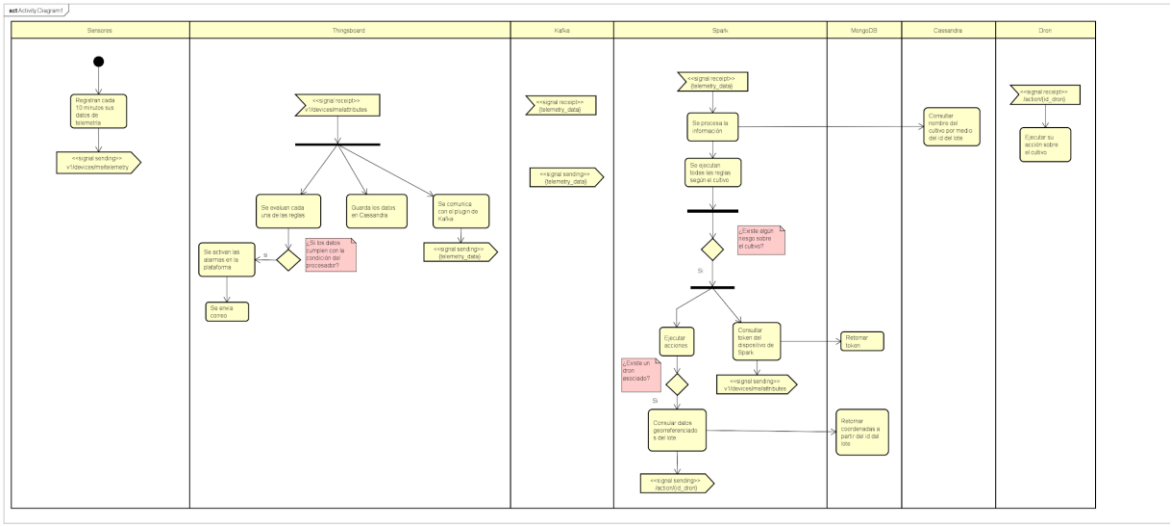
https://drive.google.com/open?id=1Hery_YBiA0YTbZRIRy5P5300DKpNPqmC

3.4.2. Vista física



https://drive.google.com/open?id=1gTSpAVq6N9eaXFMI3xWHTOpkZTZqAAKH

3.4.3. Vista dinámica



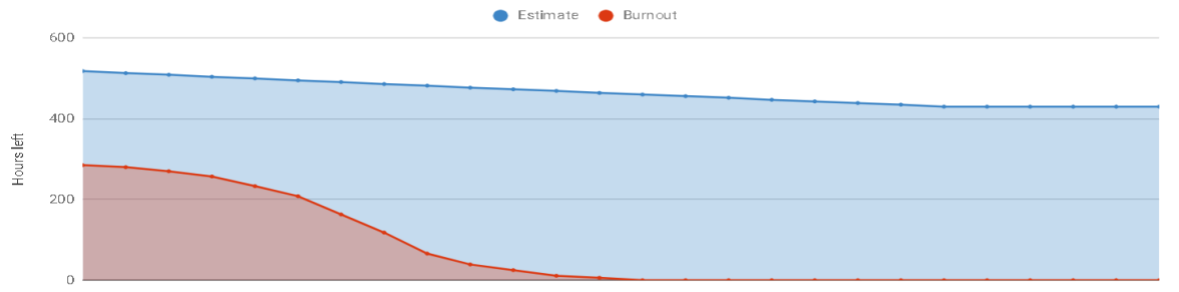
https://drive.google.com/open?id=1Il3zOYMc9TQkFXj0N32SJvFAknfKrTtl

4. Proceso de desarrollo

La extensión de la plataforma de Thingsboard llevó 6 meses y medio entre las fechas: (23 de enero del 2018 - 23 de mayo del 2018) y (30 de mayo del 2018 - 26 de julio del 2018) dividido en 2 sprints y se especificarán a continuación teniendo en cuenta Trello para especificar las tareas (https://trello.com/b/V6wD9VEX):

Para el primer Sprint se definieron las historias de usuario junto con sus tareas y el responsable donde cada uno debió registrar el tiempo que dedicó para completar dicha tarea.

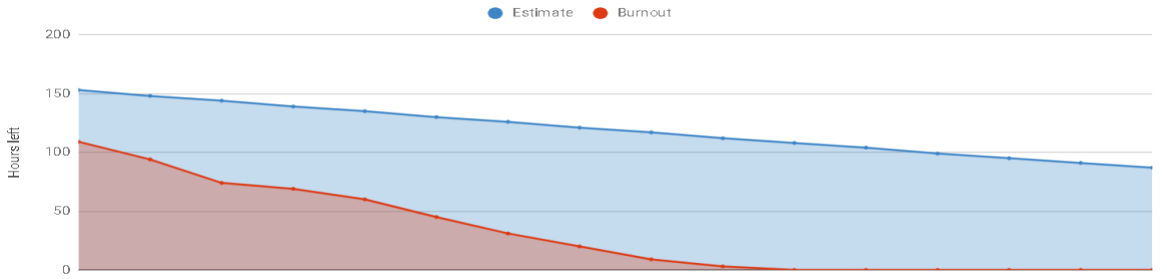
User Story	Task	Responsible	Time (estimated)	Time (spent)	Time (left)	30	31	1	2	3	4	5	6	7	8	9	10	11
Administrar un cultivo de un lote	Asociar cultivo a lote y agregar detalles	German	13	13	0		2	4		2	3	2						
Administrar un cultivo de un lote	Indicar terminación del cultivo de un lote	German	13	13	0							3	5	5				
Administrar un cultivo de un lote	Registrar visitas técnicas y acciones que se realizaron en un cultivo	German	20	20	0							5	5	3	5	2		
Registrar finca	Ubicación (Polígono)	German	20	16	4							5	6	3	2			
Registrar finca	Descripción de ubicación	Cristian	3	5	0		2	2	1									
Registrar finca	Extensión finca	Cristian	5	5	0		2		1	2								
Registrar finca	Destinación	German	13	13	0		2	5	3	3								
Registrar finca	Lista de lotes con sus cultivos	Carlos	8	7	1								3	4				
Registrar finca	Administrar tecnologías	Cristian	13	11	2							5	6					
Registrar finca	Descripción del hogar y personas	Cristian	13	12	1						3	2	2	3			2	
Registrar finca	Climatología local	German	40	40	0		2				10	4	6	3	5	10		
Registrar finca	Orografía	Cristian	3	3	0						3							
Registrar finca	Medios de acceso a la finca	Cristian	3	3	0						1	2						
Registrar finca	Estado de la carretera	Cristian	3	3	0					1	2							
Registrar finca	Distancia cabeceras municipales	Cristian	20	3	17				3									
Registrar finca	Transporte para producción	Cristian	3	3	0													3
Registrar finca	Servicios públicos	Cristian	8	8	0			2							1	2	3	
Registrar finca	Puntos de toma de agua	German	8	8	0				5	2	1							
Registrar finca	Sistemas de riego	German	3	3	0				3									
Registrar finca	Crear dashboard dinámicamente	German	0	0	0													
Asociar lote a una finca	Ubicar	Carlos	20	15	5					5	6	4						
Asociar lote a una finca	texto descriptivo	Cristian	3	3	0													3
Asociar lote a una finca	extension	German	5	5	0	5												
Asociar lote a una finca	tipo de suelo	German	20	20	0				3	5	5	3	4					
Asociar lote a una finca	historial de cultivos	German	20	20	0				5	5	6	4						
Registrar sensor a lote	Ubicación (Punto)	Carlos	13	0	13													
Registrar sensor a lote	Historial Telemetría	German	13	0	13													
Configurar Dashboard	Mostrar mapa según dashboard de finca	Carlos	40	0	40													
Configurar Dashboard	Mostrar lotes según finca	German	20	0	20													
Configurar Dashboard	(mapa) Dispositivos sobre lotes	Cristian	13	10	3								3	6	1			
Configurar Dashboard	Color del dispositivo según su alarma	Cristian	20	11	9							6	5					
	TOTAL		526	273	255	0	10	13	16	10	23	28	33	21	13	14	5	3
	Daily burnout		438			5	4	5	4	5	4	5	4	5	4	4	5	4
	Total time left (from estimate)				526	521	517	512	508	503	499	494	490	485	481	477	472	468
	Total time left (from spent)				273	268	258	245	221	196	156	111	66	39	25	11	6	0



Para el segundo sprint el tiempo fué más corto debido al periodo intersemestral que comprenden 8 semanas a comparación del primer sprint que fueron 16 semanas pero la duración y el trabajo invertido ué el mismo como si fuera un semestre normal. Dentro de las tareas realizadas está la implementación de tareas que faltaron dentro del sprint 1 y son necesarias para la continuidad del proyecto de acuerdo con la integración del trabajo con el departamento de ingeniería electrónica.

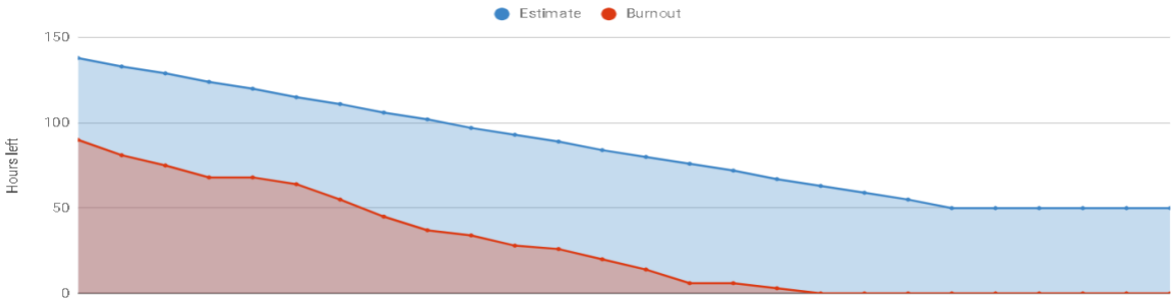
Tareas faltantes del sprint 1 para el 2 sprint

User Story	Task	Responsible	Time (estimated)	Time (spent)	Time (left)	30	31	1	2	3	4	5	6	7	8
Administrar un cultivo de un lote	Buenas prácticas	Cristian	13	9	4	4	5								
Registrar finca	Foto de fachada	Carlos	13	9	4					4	5				
Registrar sensor a lote	Ubicación (Punto)	Carlos	13	14	0					6	5	3			
Configurar Dashboard	Mostrar mapa según dashboard de finca	Carlos	20	21	0	11	10								
Configurar Dashboard	Mostrar lotes según finca	German	20	8	12			5	3						
Configurar Dashboard	(mapa) Dispositivos sobre lotes	Cristian	6	10	0							6	3		1
Configurar Dashboard	Mostrar widget asociado a dispositivo sobre mapa	German	20	0	20										
Configurar Dashboard	Color del dispositivo según su alarma	Cristian	20	16	4								8	6	2
MongoDB	Dispositivo y poligono Contenido	Carlos	20	17	3				6	5	4	2			
	Dispositivos Spark Automáticamente	German	5	5	0		5								
Configuración Thingsboard	Mismo Tamaño Ventana	Cristian	3	0	0										



Tareas que hacen parte del sprint 2

User Story	Task	Responsible	Time (estimated)	Time (spent)	Time (left)	30	31	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Etiqueta	Barra de tiempo	German	20	6	14	3		3														
Etiqueta	Mostrar datos correspondientes según fecha	Cristian	13	16	0	6	6	4														
Etiqueta	Mostrar Imagen	Cristian	40	29	11										4	2	6	3	8		3	3
Etiqueta	Etiquetar	Carlos	14	11	3						4	5	2									
Etiqueta	Generar Polígono	German	5	0	5																	
Etiqueta	Cambiar tipo de imagen	German	13	0	13																	
Fotos tomadas por drones	Subir fotos y que se agrupen	Carlos	20	16	4							5	6	3	2							
Gráficas telemetria	Mostrar 4 gráficas de los valores de telemetría between a una fecha	Carlos	13	12	1			4	5									3				



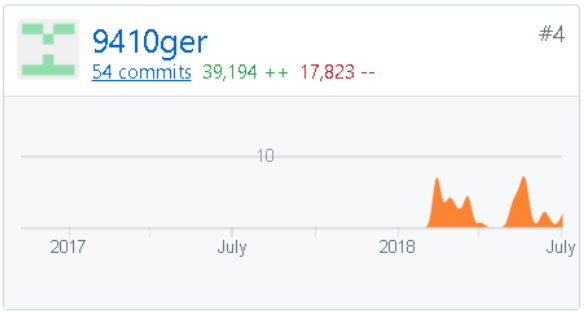
Para detallar la contribución que cada integrante del equipo realizó al proyecto de Thingsboard, sobre el Repositorio oficial LIS-Laboratorio de Ingeniería de Software (Software Engineering Laboratory - Official Repository) - Escuela Colombiana de Ingeniería.

La contribución que se va a detallar llega hasta el 22 de julio donde fueron los últimos arreglos que se realizaron para la presentación de la prueba de concepto.
El repositorio de Thingsboard y el trabajo de Spark son los siguientes:

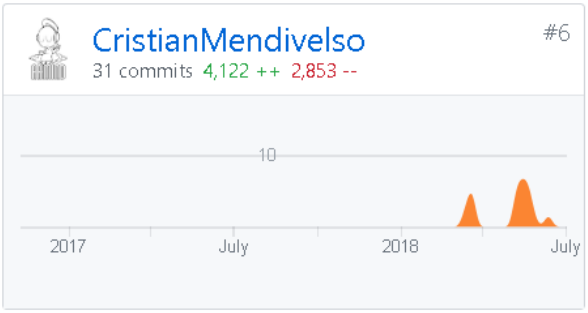
https://github.com/LIS-ECI/thingsboard

https://github.com/LIS-ECI/thingsboard-spark-backend

Integrantes:
Germán Andrés López Pacheco



Cristian Fernando Mendivelso Sanabria



Carlos Alberto Ramírez Otero

5. Liberación

5.1. Requerimientos del sistema

Requerimientos Hardware:

Para cualquiera de los dos sistemas operativos se recomienda una máquina con las siguientes características:

Procesador: Multicore (Distribución de tareas por Spark)

RAM: 2GB de RAM (4GB recomendables)

Disco Duro: Adicionales al sistema operativo (10GB disponibles)

Periféricos: Monitor, teclado, mouse (adicional), conectado a una red con otros equipos para clúster si se desea tener bases de datos remotas y nodos de Spark para trabajo en paralelo.

Sistema Operativo: Thingsboard puede ser instalado en las versiones de Windows 10/8.1/7 y Linux(Ubuntu 16.04) ya que fueron las versiones usadas en el proyecto y dentro del manual de instalación se realiza el debido procedimiento para que funcione.

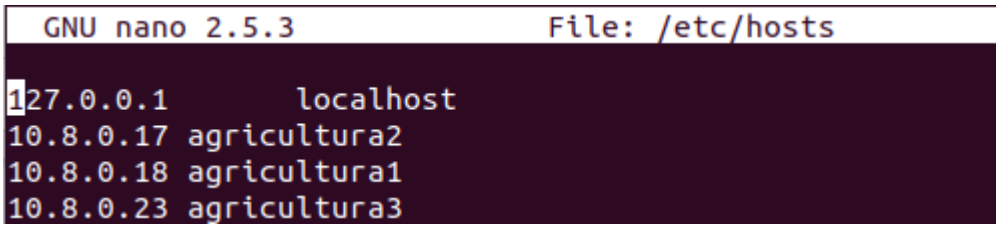
Librerías: Git, JDK 8, Maven, Cassandra (DataStax Community Edition v3.0.9), MongoDB 3.2.19

La implementación de la solución se desarrolló en 3 equipos:

# Equipo	Hostname	IP	Herramientas
1	agricultura1	10.8.0.18	Kafka Instancia Thingsboard Cassandra Nodo trabajador de Spark
2	agricultura2	10.8.0.17	Redis Nodo master y trabajador de Spark
3	agricultura3	10.8.0.23	MongoDB Nodo trabajador de Spark

Es válido aclarar que, en el archivo de configuración, todos los equipos deben conocer los hostname de los demás (para efectos del clúster de spark)

para configurar esto puede hacerlo a través del comando:
sudo nano /etc/hosts
De forma que quede de esta manera:



127.0.0.1 localhost

10.8.0.17 agricultura2

10.8.0.18 agricultura1

10.8.0.23 agricultura3

5.2. Manual de instalación

En el repositorio del proyecto https://github.com/LIS-ECI/thingsboard se encuentra el manual de instalación para poder usar Thingsboard y todos los componentes añadidos que hacen parte de la contribución realizada.

El link directo es el siguiente:

https://github.com/LIS-ECI/thingsboard/blob/master/Installation_Manual.md

Configurar conexión de base de datos de MongoDB en Thingsboard

En Thingsboard, se realiza la conexión a la base de datos MongoDB a un servidor que se encuentra en la Escuela Colombiana de Ingeniería y por lo tanto cualquier equipo debe estar en la misma red o tener instalada la VPN que comunica el equipo con este servidor desde cualquier red.

Para poder cambiar la IP a la que Thingsboard deberá conectarse para usar MongoDB en el equipo instalado (local o remoto), deberá acceder a la siguiente ruta del proyecto, sirve para cualquiera de los dos sistemas operativos:

thingsboard/dao/src/main/resources/config.properties

```
10 lines (8 sloc)  273 Bytes
```

```
1  #MongoDB Configuration
2  #Change each value with the corresponding mongodb conection parameter
3  #@host -> MongoDB server IP
4  #@port -> Conection Port to conect MongoDB
5  #@database -> MongoDB database name
6
7  mongodb.host = 10.8.0.23
8  mongodb.port = 27017
9  mongodb.database = prueba
```

El cual, se describirán los siguientes parámetros que se deberán cambiar para acceder al servicio de Mongo:

Mongodb.host: IP del equipo que posee el servicio remoto de MongoDB o localhost en caso de tener el servicio sobre el mismo equipo en el que se está ejecutando MongoDB.

Mongodb.port: Puerto habilitado para la comunicación con el servicio de MongoDB.

Mongodb.database: Nombre de la base de datos a la que accederá Thingsboard para persistir su información.

5.3. Manual de usuario

Índice:

Botones básicos:

 Botón de creación

 Botón de edición

 Botón de creación y cancelación respectivamente

5.3.1 Ingresar al sistema:

1. Entrar a la página principal de Thingsboard: localhost:8080 (en el caso de que el programa se esté ejecutando localmente)
2. para ingresar al sistema:

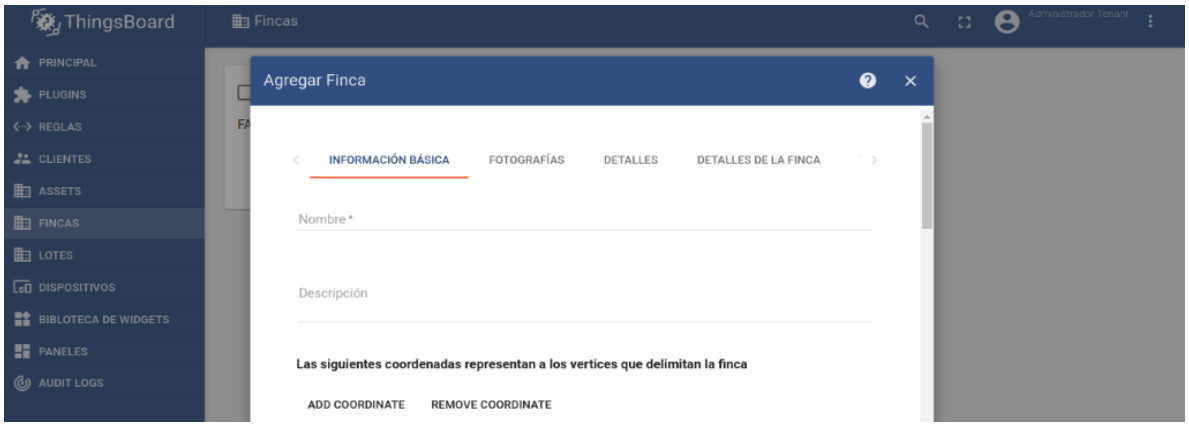
usuario: tenant@thingsboard.org **contraseña:** tenant

5.3.2 Crear una finca:

URL Video de apoyo:
https://drive.google.com/file/d/1rItMSD0jlLe6NpnVHGlPjWJ2sNuJzqUG/view?usp=sharing

1. Ir a la pestaña de Fincas
2. Oprimir el botón de creación

A continuación, saldrá una venta como la siguiente:

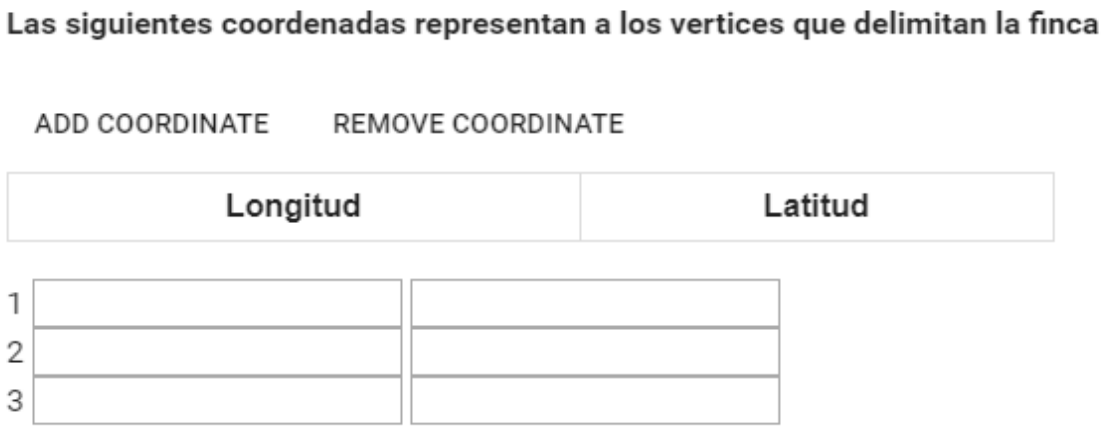


En esta ventana se puede llenar información básica como el nombre de la finca, la descripción, y puede añadir las coordenadas de los vértices que delimitan la finca para que se pueda posteriormente visualizar en un mapa.

Estas coordenadas se pueden obtener en el documento de Registro de propiedad de la finca, o utilizando la herramienta http://geojson.io y dibujando el polígono de la finca (Tal y como se muestra en el video)

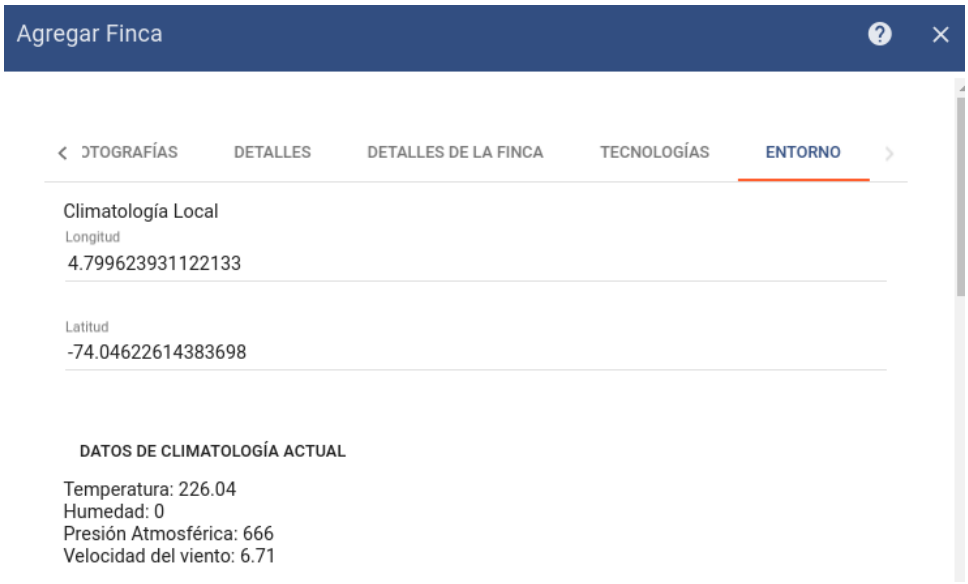
Para esto posee dos botones Añadir coordenada o eliminar coordenada, según sea necesario, y también una descripción de la localización, y el área total, tanto en hectáreas como en fanegadas.

3. Llenar la información
4. Una vez ubicadas las coordenadas debe oprimir en ubicar finca



5. Es recomendable llenar toda la información de la finca navegando por las pestañas, sin embargo, puede llenarla posteriormente a su creación

La pestaña de fotografías se recomienda diligenciar luego de crear la finca.
6. En la pestaña detalles, puede llenar información sobre la destinación de la finca, detalles de uso, seleccionar los servicios públicos con los que cuenta la finca, el tipo de transporte para la producción, y puede crear puntos de agua a partir de un identificar y el número de resolución
7. En la pestaña detalles de la finca, puede completar información sobre la finca y las personas
8. En la pestaña de tecnologías, actualmente puede añadir sistemas de riego con un nombre y una descripción
9. En la pestaña entorno puede obtener los datos de la climatología local, debe poner los datos de la longitud y la latitud y al oprimir el botón los datos de climatología local

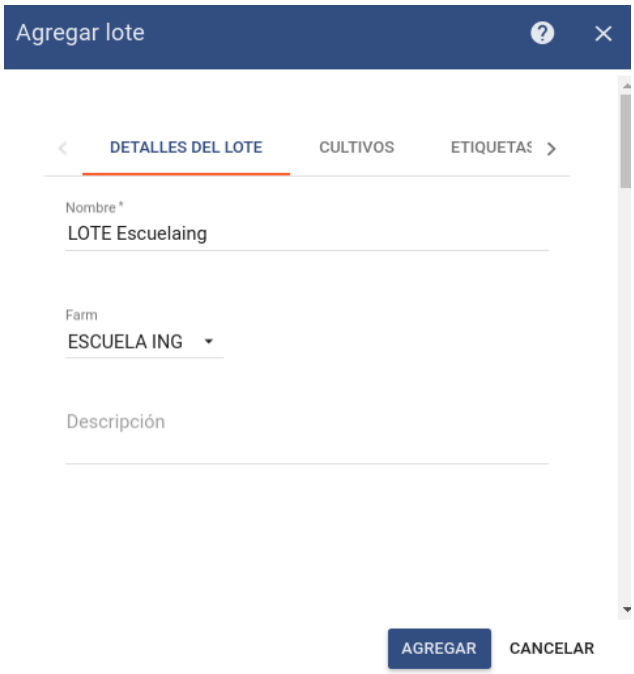


5.3.3 Crear lote y un cultivo

Url Video de apoyo:

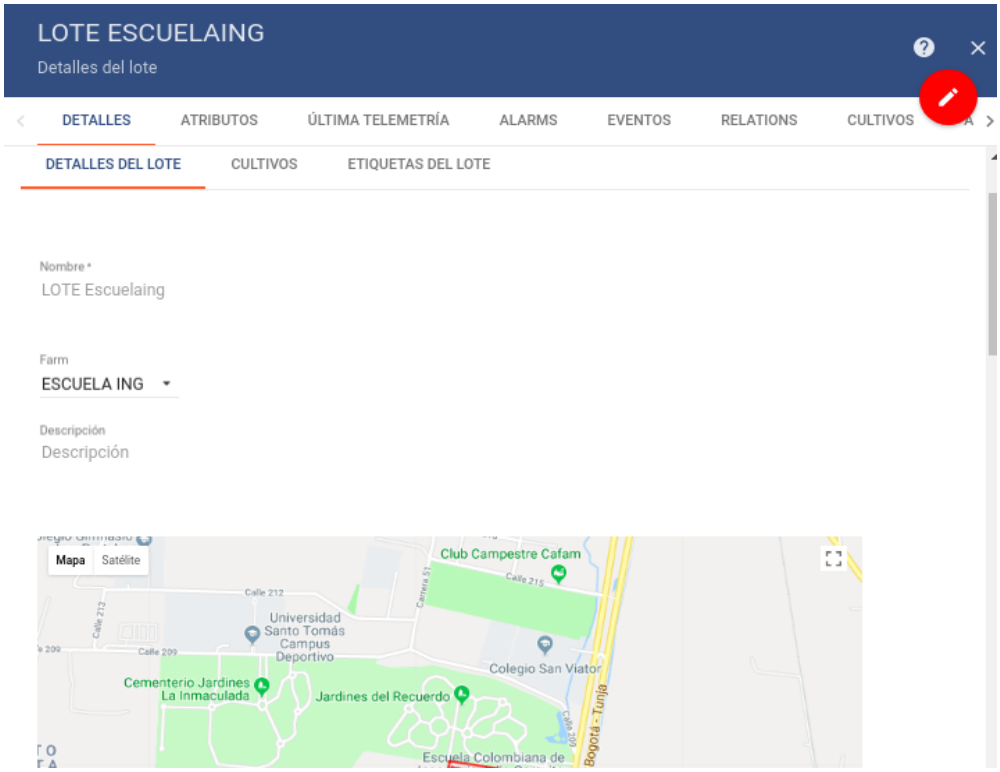
https://drive.google.com/open?id=1TnERW7XON6BkN8KiLIn6eG5xd6Ef2ZnC

1. Ir a la pestaña de lotes, y oprimir en el botón de creación
2. Saldrá una ventana como la siguiente:

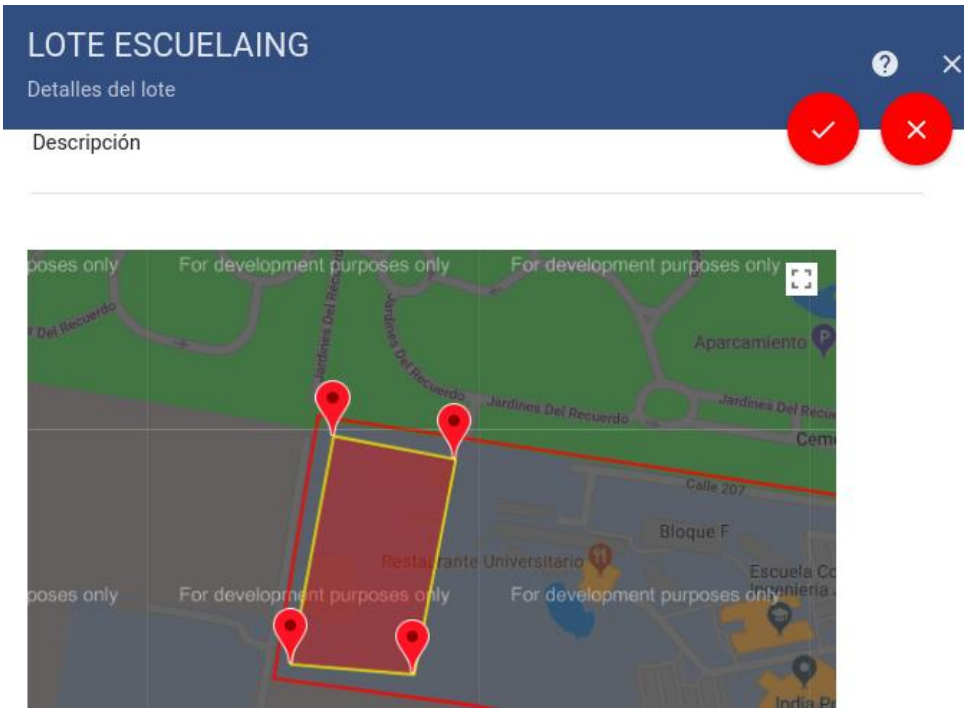


3. Se recomienda que, a la hora de crear el lote, sólo se llene la información de la pestaña detalles del lote

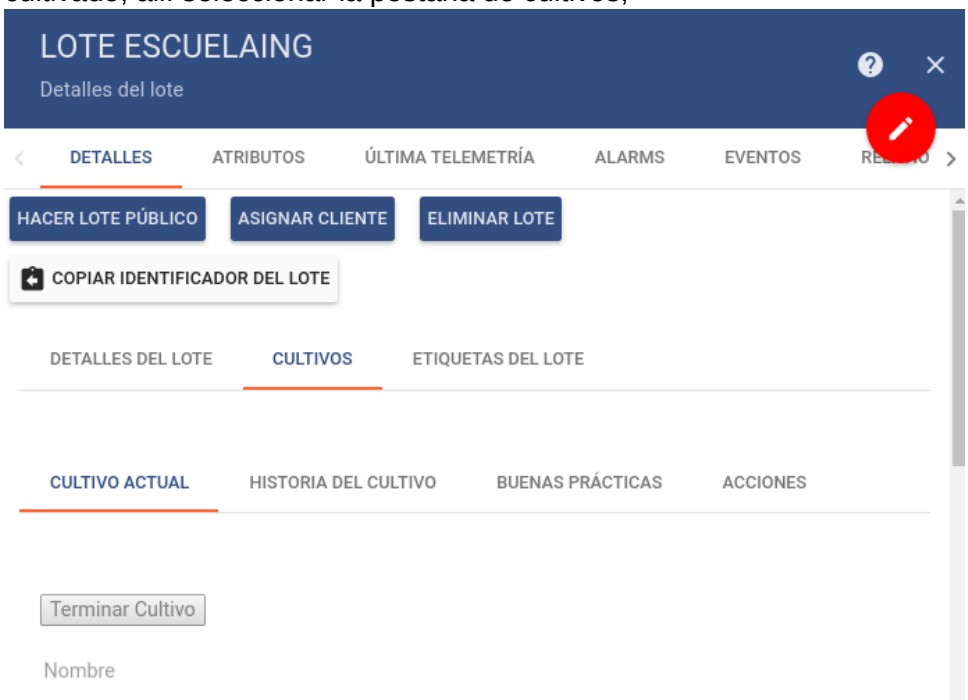
4. Una vez creado, para editar un lote debe darle click en el botón de edición



5. Allí puede dibujar el polígono del lote, dando clic en los vértices delimitando el lote, para guardar el polígono debe darle click en el botón de confirmación



6. Para crear un cultivo debe ir a la pestaña de lotes, y seleccionar el lote en dónde es cultivado, allí seleccionar la pestaña de cultivos,



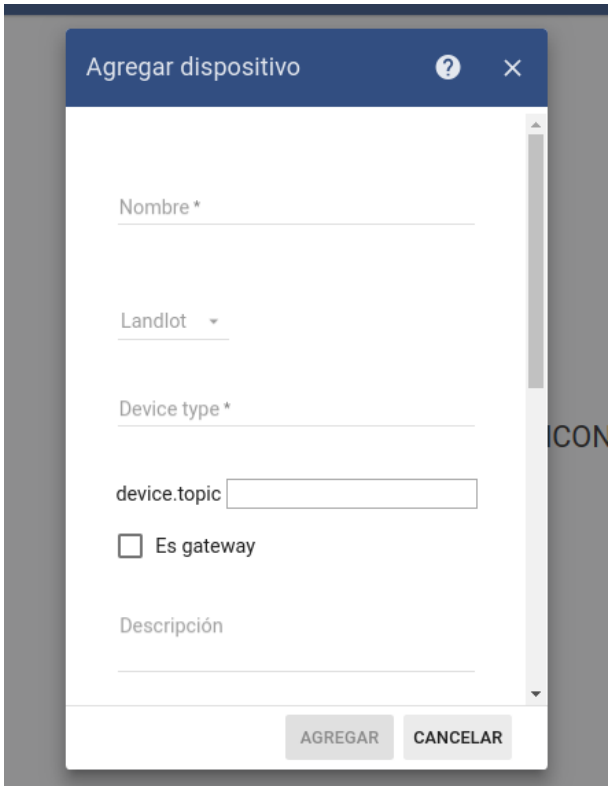
7. Debe darle clic en el botón de edición y asignar un nombre al cultivo entre otra información, y para guardarlo debe darle clic en el botón de confirmación (no en terminar cultivo, esta opción se utiliza para darle fin al cultivo)

8. Puede llenar información de los cultivos con el historial, buenas prácticas, acciones, Para este ejemplo crearemos un cultivo de papa, para esto su nombre será Papa.

Creación de dispositivos:

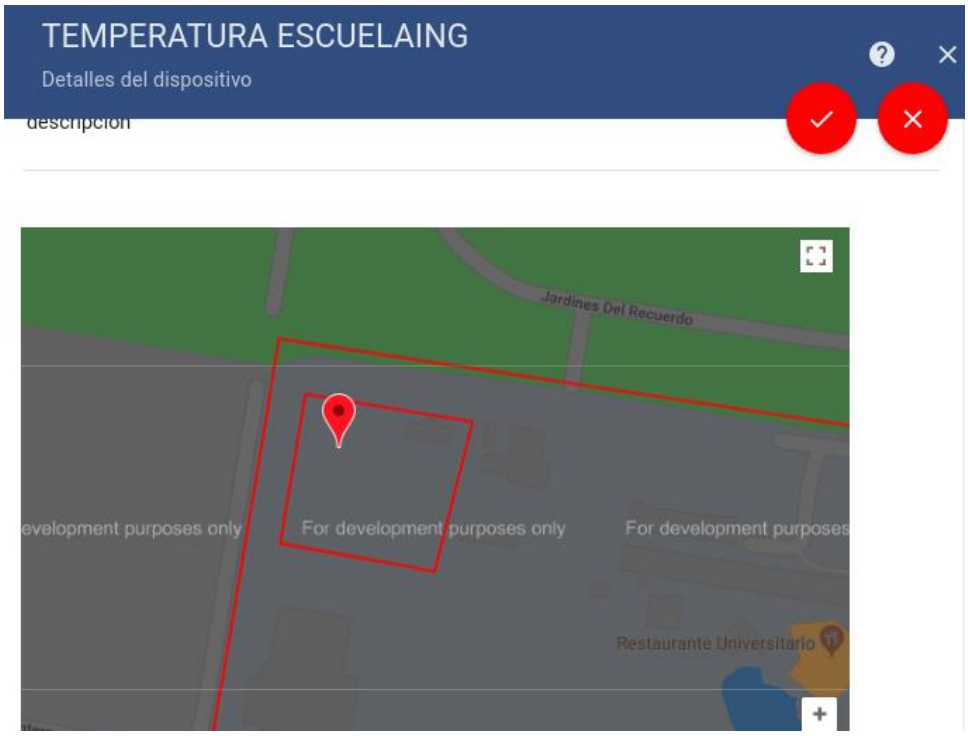
Url Video de Apoyo: https://drive.google.com/open?id=1zdQhsiD-8gCq7LMV-80gNCg9-bowo3lM

1. Debe ir a la pestaña de dispositivos, y dar clic en el botón de agregar, posteriormente aparecerá una ventana como esta:



donde puede llenar información como el nombre del dispositivo, en este caso se llamará “Sensor de temperatura”, debe escoger el lote asociado (el que acabamos de crear), y escoger el tipo de dispositivo. Puede escoger el tipo de dispositivo por defecto: default, en este caso se escribirá “temperature” cómo Device type, otra opción (que será analizada más adelante, cuando es un dispositivo de tipo spark- solo en ese caso debe llenar el device.topic).

2. Una vez creado puede editarlo seleccionandolo, y dando clic en el botón de edición y se procede a ubicar en el mapa el dispositivo creado dándole clic sobre el mapa y guardando la ubicación dando clic en el botón de confirmación.



Para el ejemplo también debe crear un Dispositivo de humedad, cuyo tipo es humidity, y otro dispositivo de Intensidad de la luz cuyo tipo es light, ambos ubicados en el lote creado previamente.

Configuración para correr un ejemplo Con Kafka y Apache Spark

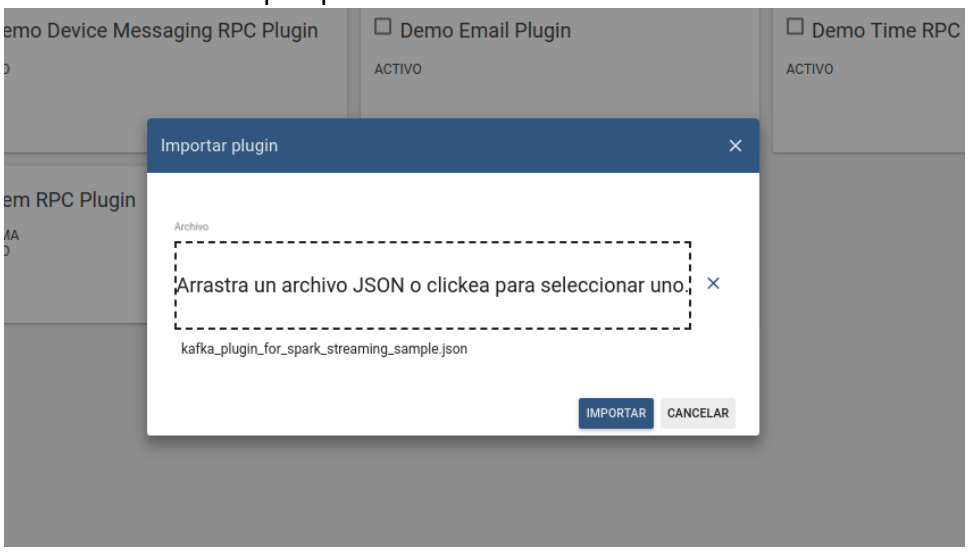
5.3.5 Creación plugin kafka:

Este plugin de Kafka, se crea solamente una vez, ya que es usado posteriormente por las reglas.
Url video de apoyo:
https://drive.google.com/open?id=1xa67WVH6vYPjTopE_eolt4pRxjs9QwWn

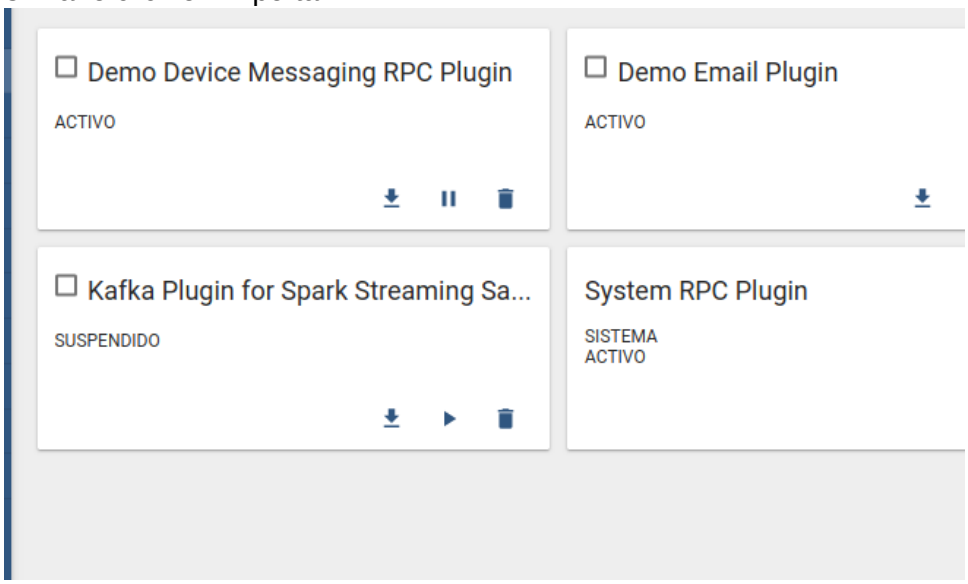
1. Click en la opción de plugins en el menú, y en el botón de creación
2. Click en la opción de importar, a continuación debe crear un archivo .json (con un editor de texto) que contenga:

{ "apiToken": "kafka-spark-streaming-sample", "name": "Kafka Plugin for Spark Streaming Sample", "clazz": "org.thingsboard.server.extensions.kafka.plugin.KafkaPlugin", "publicAccess": false, "state": "SUSPENDED", "configuration": { "bootstrapServers": "localhost:9092", "batchSize": 16384, "bufferMemory": 33554432, "acks": -1, "keySerializer": "org.apache.kafka.common.serialization.StringSerializer", "valueSerializer": "org.apache.kafka.common.serialization.StringSerializer" }, "additionalInfo": null}

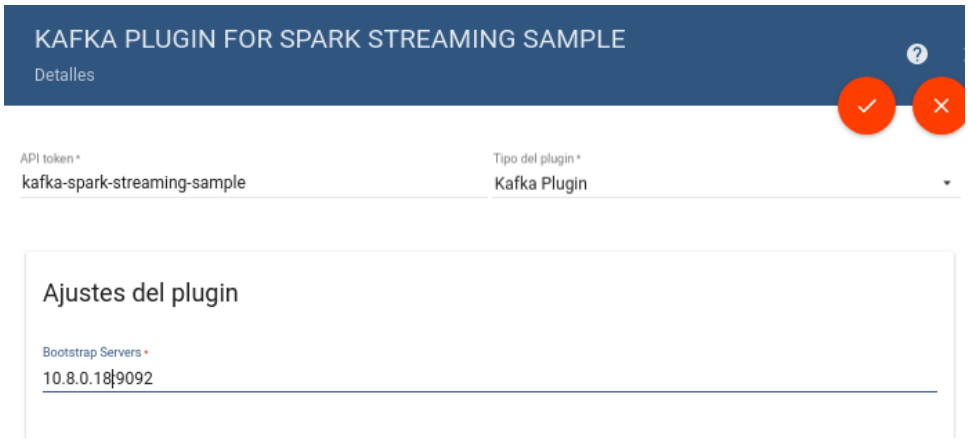
Y anexar de forma que quede así:



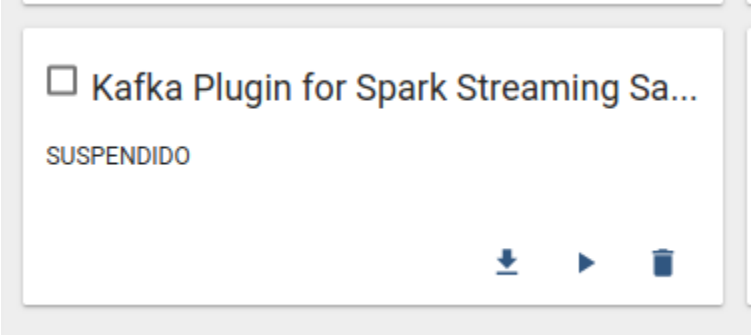
3. Darle click en importar



4. Una vez creado podemos editarlo dando click sobre él y sobre el botón de edición, como en este ejemplo el servidor kafka está corriendo en el servidor 10.8.0.18 se procede a editar el bootstrap servers con la dirección de la siguiente manera y guardar el cambio dándole click en el botón de confirmación



5. Posteriormente se activa el plugin dándole click en el botón de play



5.3.6 Creación de reglas:

Url Video de apoyo:
https://drive.google.com/open?id=1IPGnc5cOLmGjiPKDdEVFPSU5Oo5eNf7a

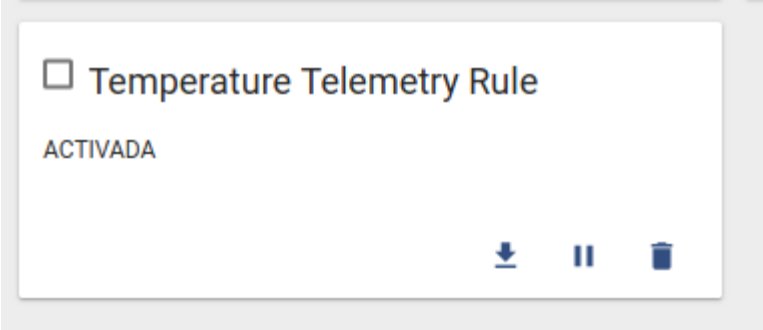
A continuación, se procede a la creación de una regla para que los datos que se envien al dispositivo, para esto se debe tener creado un plugin de Kafka.

1. Click en la opción del menú derecho de reglas, y en el botón de creación de nueva regla, y seleccionar importar regla
2. a continuación debe crear un archivo .json (con un editor de texto) que contenga:

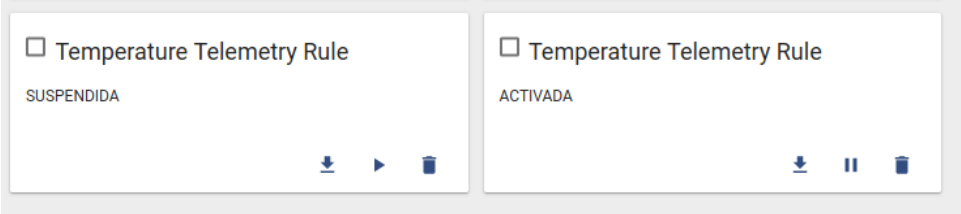
{"additionalInfo": null,"name": "Temperature Telemetry Rule","state": "ACTIVE","weight": 0,"pluginToken": "kafka-spark-streaming-sample","filters": [{"configuration": {"messageTypes": ["POST_TELEMETRY"]},"name": "MsgTypeFilter","clazz": "org.thingsboard.server.extensions.core.filter.MsgTypeFilter"},{"configuration": {"filter": "typeof temperature !== 'undefined'"},"name": "TelemetryFilter","clazz": "org.thingsboard.server.extensions.core.filter.DeviceTelemetryFilter"}],"processor": null,"action": {"configuration": {"sync": true,"topic": "temperature","template": "{\"deviceId\":\"$ss.get('deviceId')\", \"temperature\":$temperature.valueAsString}"},"clazz":

"org.thingsboard.server.extensions.kafka.action.KafkaPluginAction","name":
"Temperature"}}

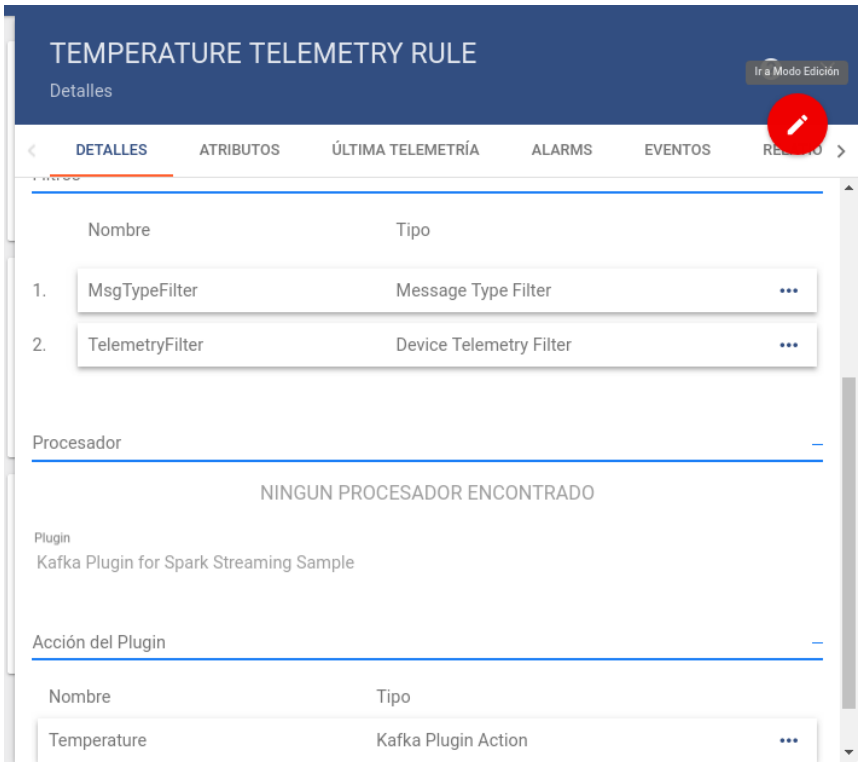
vale la pena aclarar que esta regla es la base para crear cualquier regla sobre datos de telemetría, a continuación se creará a partir de esta una regla para los datos de humedad
3. Anexar el .json tal como en el paso 20 y click en importar
4. Darle click en play para activar la regla



5. Ahora se procederá a Configurar la regla para los dispositivos de humedad usando como plantilla la regla para los dispositivos de temperatura importada anteriormente, Para esto, vamos a importar de nuevo el .json que creamos de modo que tengamos en el sistema 2 reglas, una activada y la otra suspendida

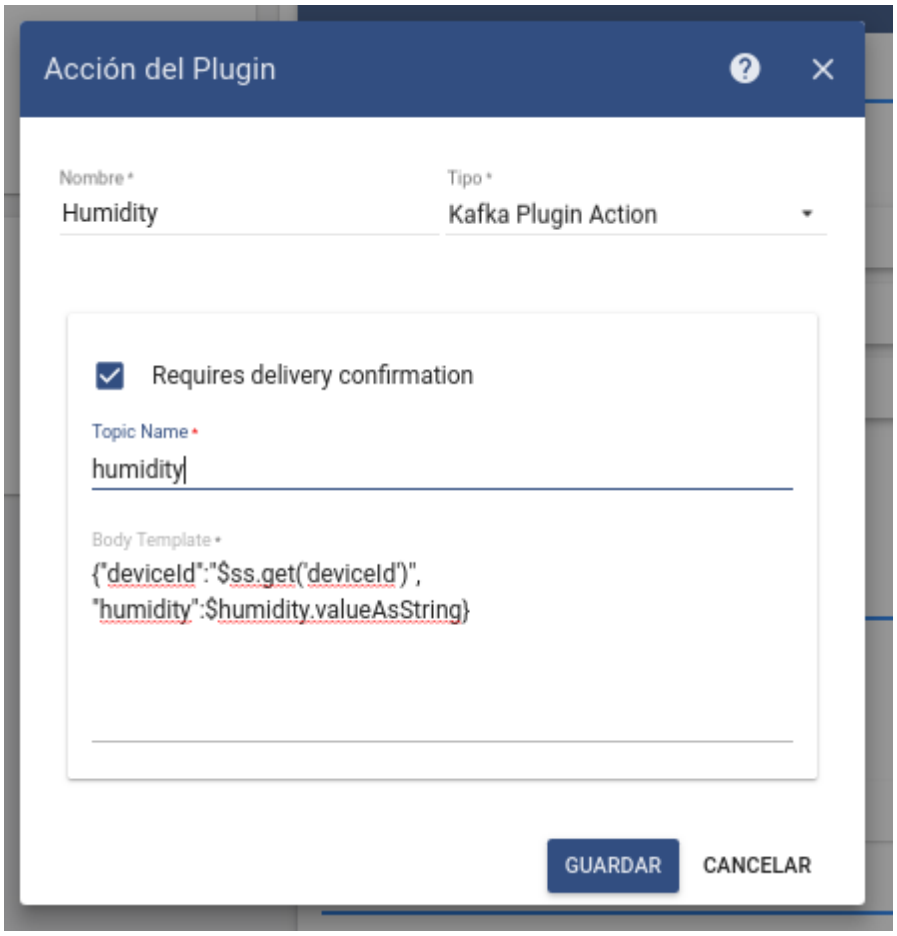


6. La regla suspendida es la que vamos a editar, para esto la seleccionamos y damos clic en el botón de edición, en la esquina superior derecha

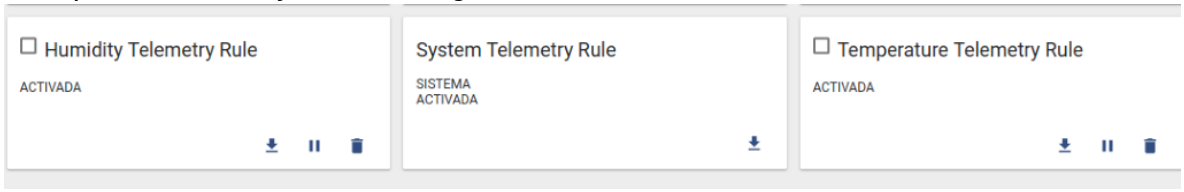


7. Cambiar el nombre de la regla por Humidity Telemetry Rule

8. Editar el filtro “TelemetryFilter” y cambiar typeof temperature !== 'undefined' por typeof humidity !== 'undefined' y darle el botón de guardar.
9. Editar la acción del plugin, cambiando el nombre por humidity, y reemplazando todo lo que diga temperature en Body Template por humidity y darle en guardar



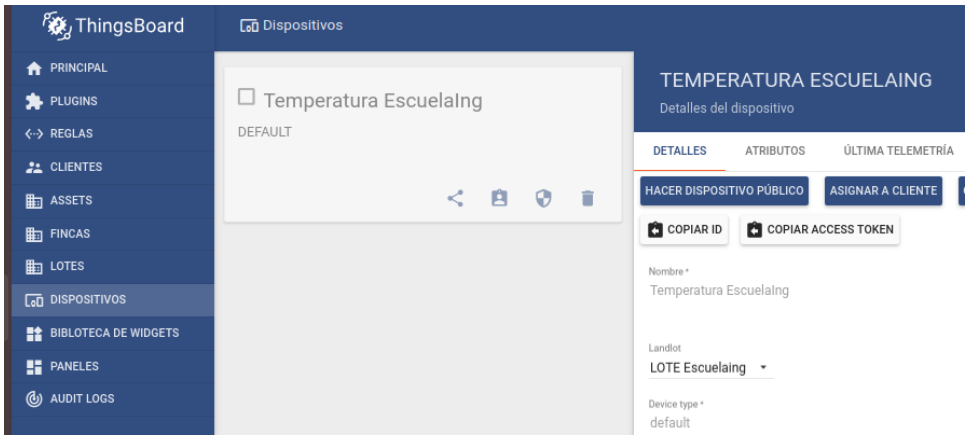
11. Aplicar cambios, y activar la regla



Como se puede ver, se pueden crear múltiples reglas para todos los tipos de datos de telemetría que se necesiten.
En este caso para la prueba de concepto se debe crear otra regla pero enfocada en la intensidad de la luz, y cuya clave será “light”

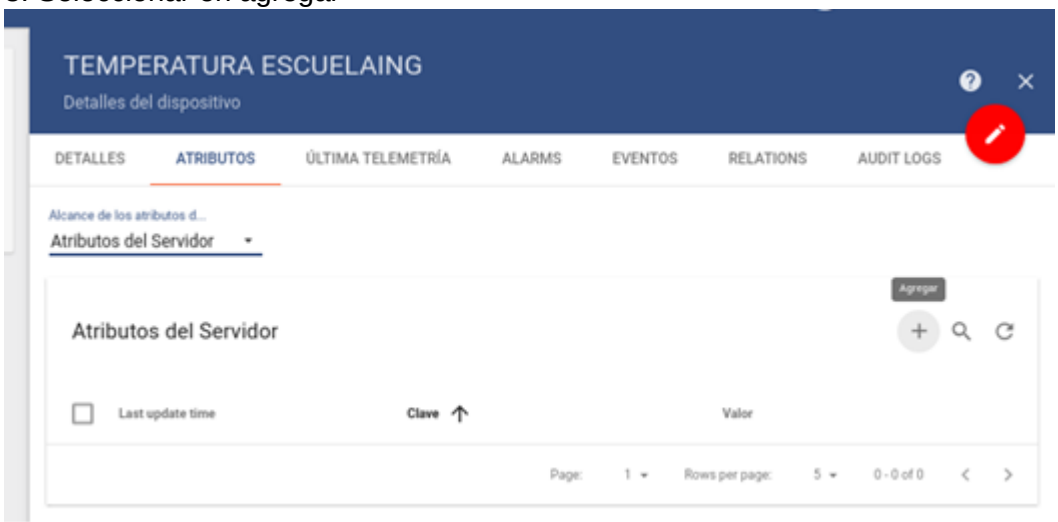
Edición de los dispositivos para que publiquen su telemetría en Kafka

Para que los mensajes de telemetría lleguen tal y como las aplicaciones de Spark lo necesita, se deben realizar las siguientes configuraciones sobre los dispositivos.
1. Ir a la pestaña de dispositivos, seleccionar el dispositivo de temperatura y copiar el Id dando clic en el botón en COPIAR ID,



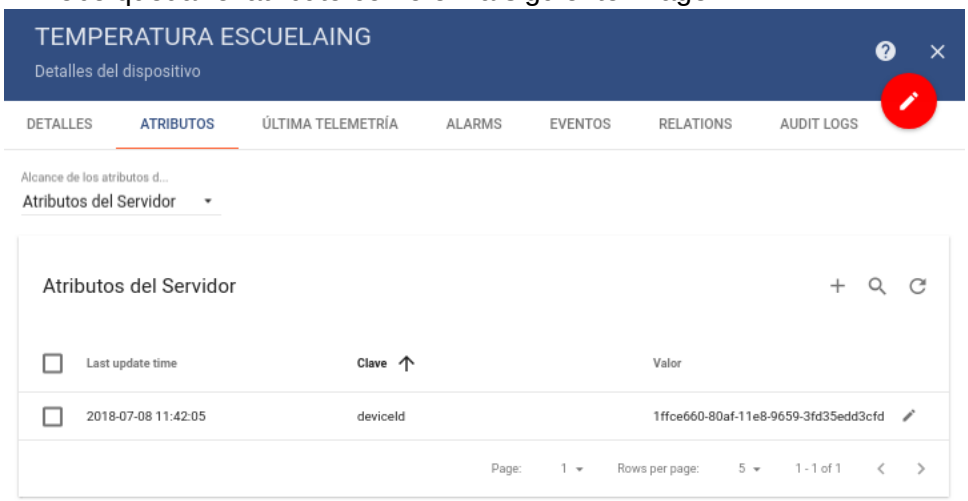
2. Ir a la pestaña de atributos, y cambiar de atributos del cliente a atributos del servidor de manera que quede como en la imagen

3. Seleccionar en agregar



Y posteriormente llenar la clave con la palabra “deviceId”, y en el valor pegamos el ID copiado anteriormente, posteriormente clic en agregar

4. Debe quedar el atributo como en la siguiente imagen

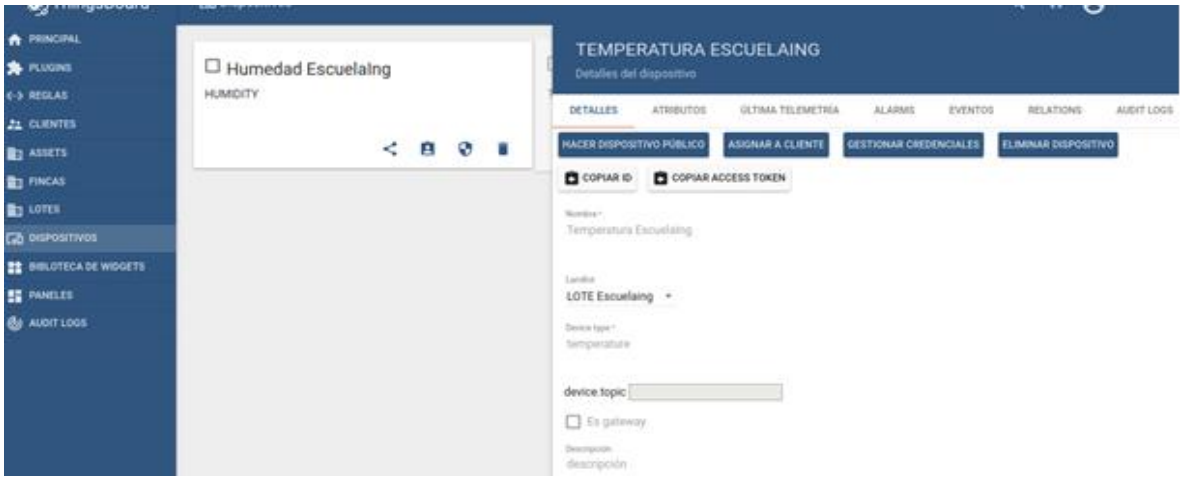


5. Agregar el atributo también para el dispositivo de humedad, y para el dispositivo de intensidad de la luz “light”.

5.3.7 Envío de datos simulados:

Url video de apoyo:
https://drive.google.com/open?id=1iKM08Ld3Xo3HbdoV2_Vgn60rljYnItOh

En Thingsboard los dispositivos son un puente para que los sensores reales pueden enviar datos, en este caso se utilizarán datos simulados por practicidad.
Se crearon 2 dispositivos, 1 de temperatura y otro de humedad, estos tienen un token con el cual los dispositivos reales pueden hacer el envío de datos, para obtener dicho token, se tiene que ir a la pestaña de dispositivos, seleccionar el dispositivo deseado y darle click en el botón de COPIAR ACCESS TOKEN



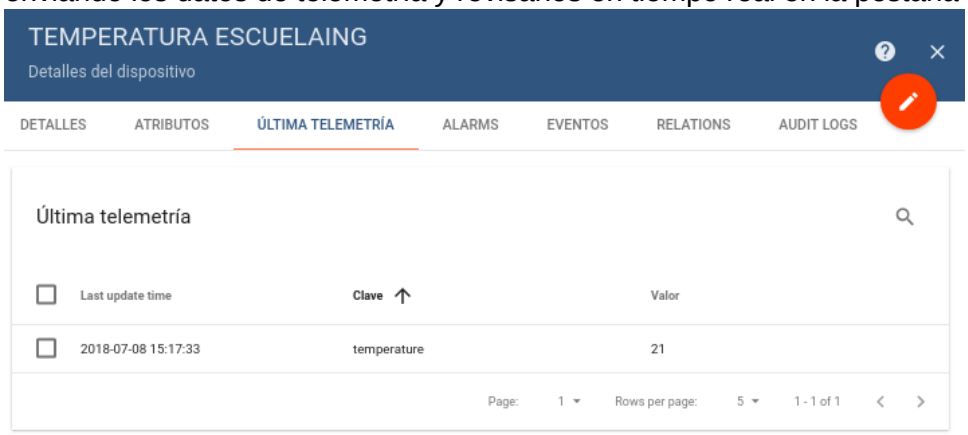
1. Para el envío de datos simulados se utilizará Python, en este caso debe guardar el siguiente código como temperature_send.py, se debe reemplazar TOKEN por el token copiado previamente manteniendo las “”, igualmente con la IP que debe ser reemplazada por la Ip del servidor en donde se esté ejecutando Thingsboard, puede editar los valores a enviar en la línea x = random.randrange(17, 23), en la cual se define el rango de los valores que se enviarán aleatoriamente

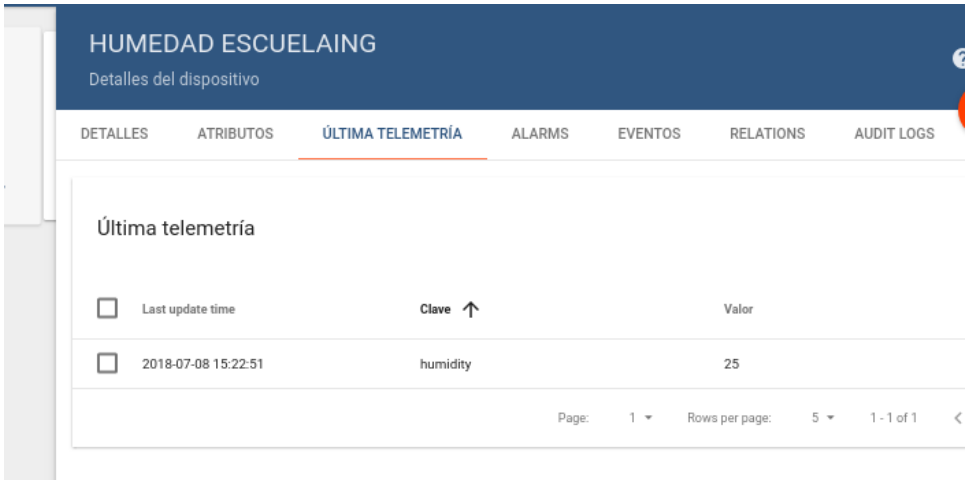
```
import paho.mqtt.client as mqtt
from time import sleep
import random

broker="test.mosquitto.org"
topic_pub='v1/devices/me/telemetry'

client = mqtt.Client()
#Debe editar el TOKEN con el Token del dispositivo al cual quiera enviar datos
client.username_pw_set("TOKEN")
#Debe editar IP a la ip del servidor de Thingsboard
client.connect('IP', 1883, 1)

while True:
    x = random.randrange(17, 23)
    print x
    msg = '{"temperature":"'+ str(x) + '"}'
    client.publish(topic_pub, msg)
    sleep(2)
```

2. Una vez guardado el archivo, se procede a ejecutar con el comando (en la carpeta dónde fue guardado)
python temperature_send.py

3. Para enviar los datos de telemetría de humedad debe cambiar en el msg temperature por humidity con su respectivo token.

4. En la pestaña de dispositivo, se puede seleccionar el dispositivo al que se le están enviando los datos de telemetría y revisarlos en tiempo real en la pestaña última telemetría

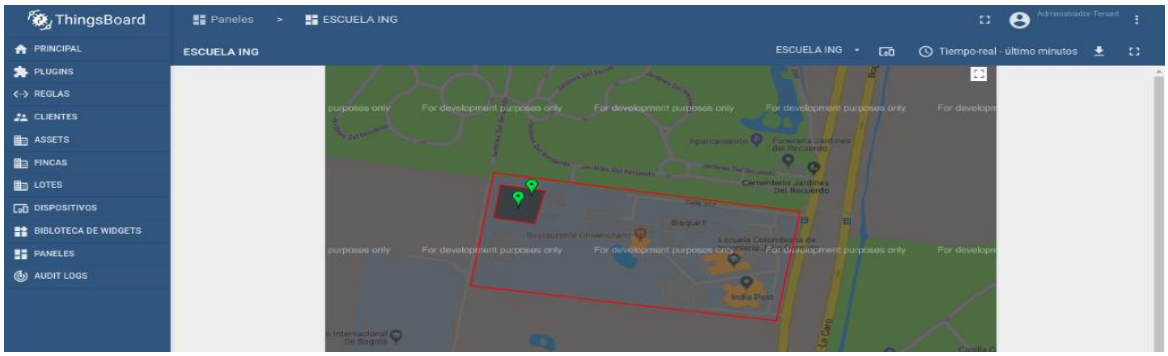




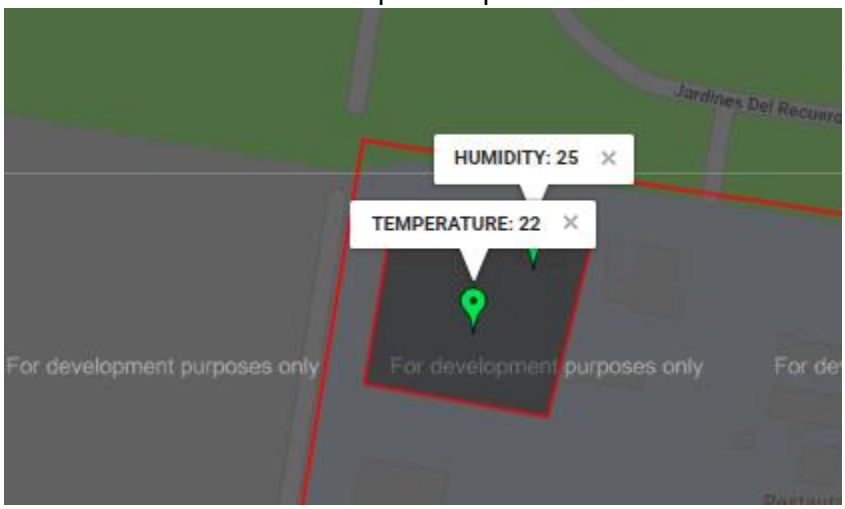
5.3.8 Revisión de paneles

1. En la pestaña de Paneles puede encontrar un panel por cada finca, al darle clic puede observar un mapa con las siguientes características:

- Se muestra el polígono que delimita la ubicación de la finca
- Se muestran todos los lotes asociados a la finca, también delimitados con un polígono
- Se muestran todos los sensores asociados a cada lote de la finca con un color dependiendo del estado del sensor, en este momento existen tres colores dependiendo la gravedad de la alarma (verde = todo está normal, naranja = alarma de warning, rojo = alarma de urgencia), esto dependerá de la configuración de las alarmas que se pueden crear utilizando Thingsboard
- Se muestra el lote con un color (azul= todo está bien, rojo= alarma de tipo Spark sobre el cultivo)



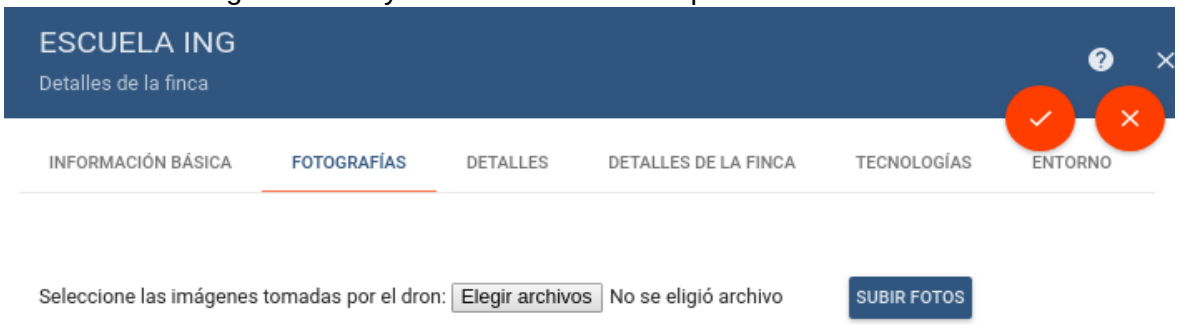
También se puede obtener el último valor de telemetría que ha llegado al dispositivo, dando clic sobre el símbolo que lo representa



5.3.9 Cargar Fotos tomadas por drones

1. Para subir al sistema las fotos tomadas por los drones, debe ir a la pestaña de fincas y seleccionar cualquier finca

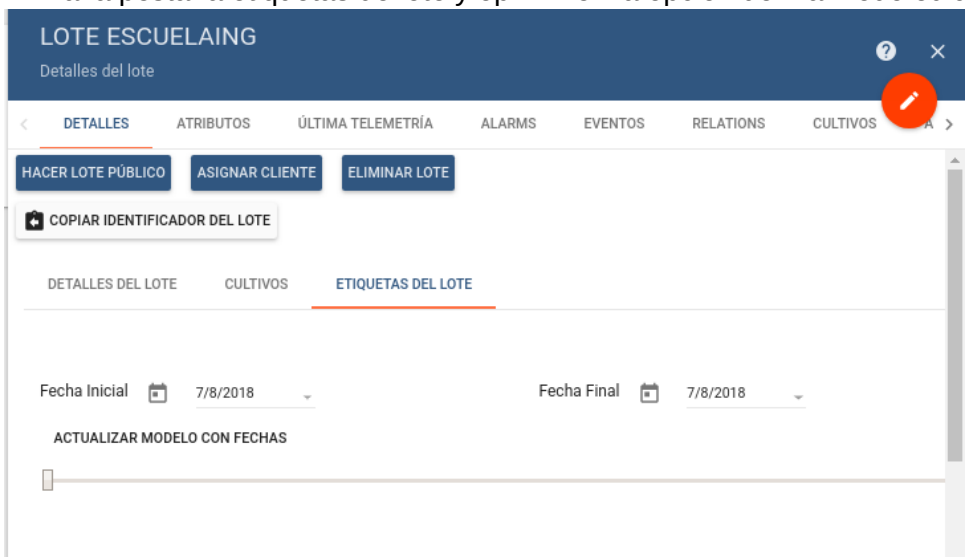
2. Debe ingresar a la pestaña de fotografías y darle en el botón naranja de ir a modo edición, ahí encontrará una opción para seleccionar un conjunto de fotos, para esto debe darle click en Elegir archivos y seleccionar las fotos que desee



Una vez haya seleccionado las fotos, debe darle click en subir fotos y esperar hasta que se hayan cargado completamente.

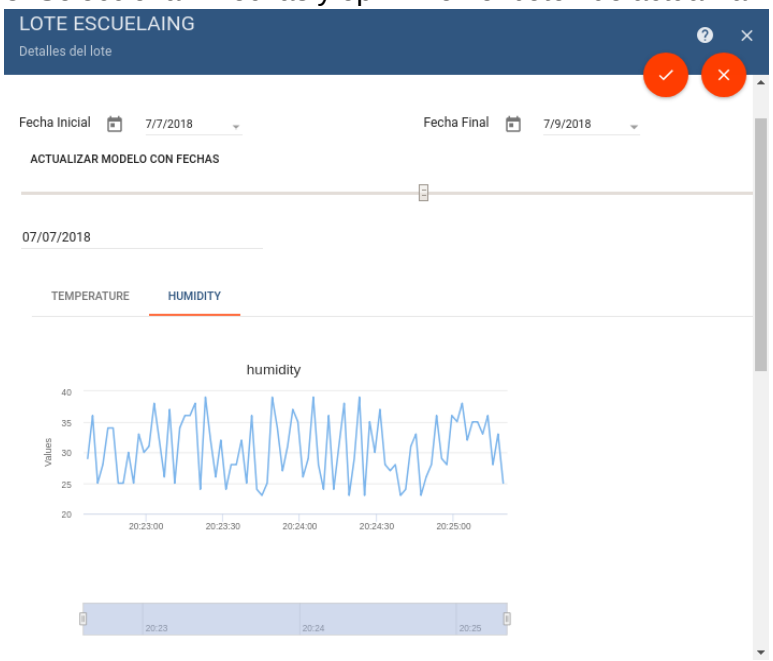
5.3.10 Elaborar Etiquetado (Para experto):

1. Para elaborar un etiquetado sobre un cultivo debe ir a la pestaña de lotes y seleccionar el lote al que se le desee hacer el etiquetado
2. Ir a la pestaña etiquetas de lote y oprimir en la opción de ir a modo edición

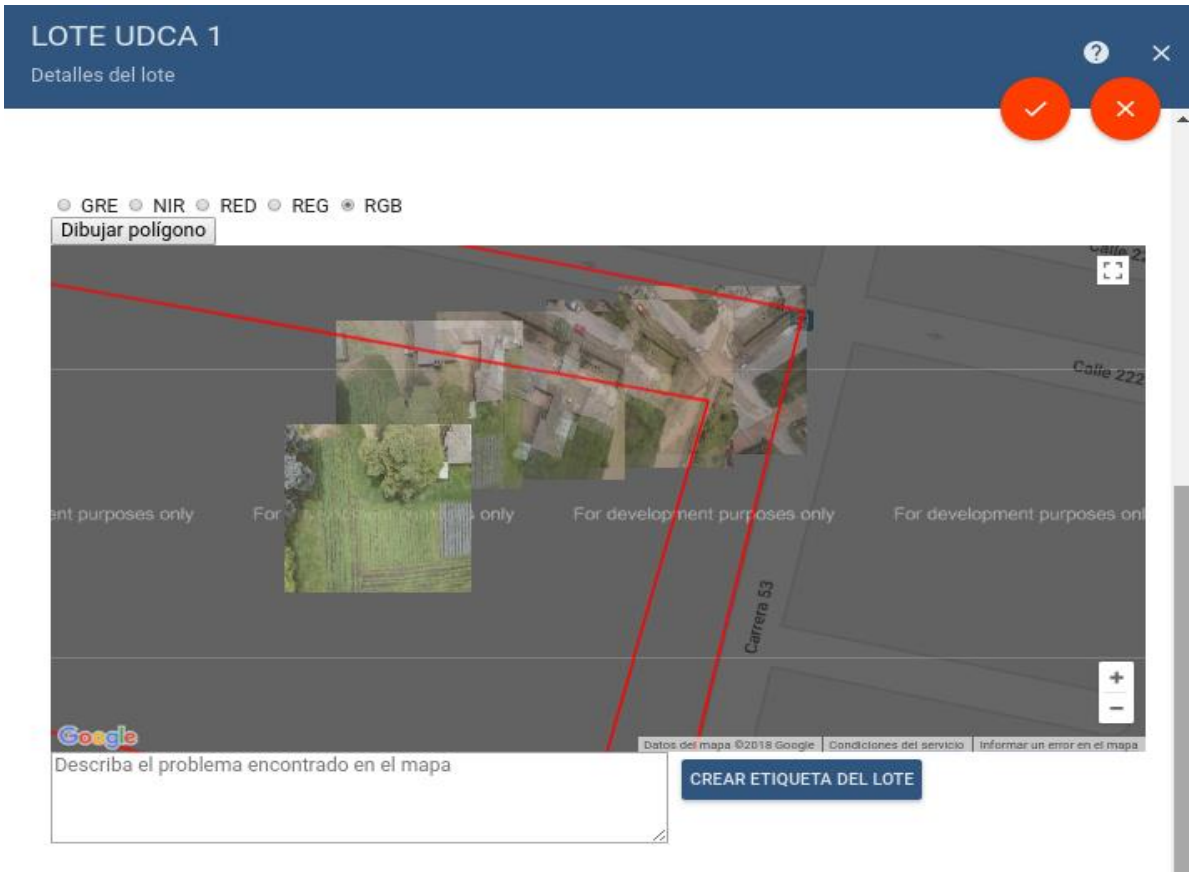


Allí puede escoger un intervalo de tiempos para poder ver todos los datos de telemetría asociados, y en caso de que haya registro de fotos en alguna fecha, puede desplazarse al día y ver las fotos sobre el mapa

3. Seleccionar 2 fechas y oprimir en el botón de actualizar modelos de fechas



4. Puede desplazarse a través de la barra de tiempo propia de las gráficas o cambiar de gráfica dependiendo los datos de telemetría que se quiera analizar

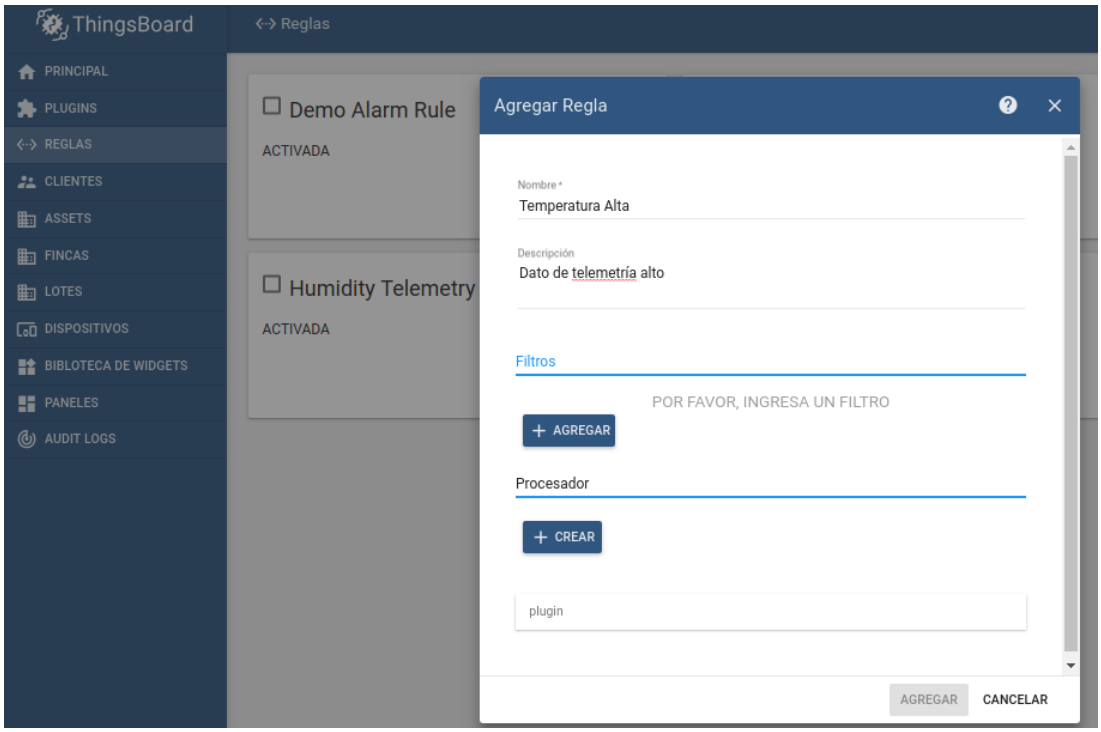


5. Puede cambiar el tipo de foto de RGB(por defecto), a GRE, NIR, RED y REG, para que se puedan hacer análisis

6. Para realizar un etiquetado debe oprimir en el botón de dibujar polígono y dibujar un polígono en la región que desee denotar, posteriormente describir el problema presentado y oprimir el botón de crear etiqueta del lote

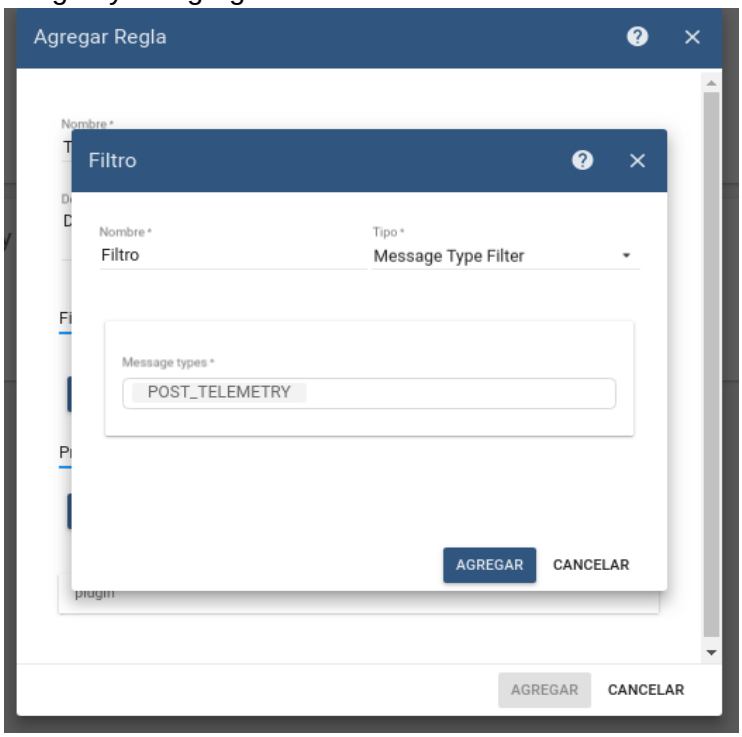
5.3.11 Crear Alertas sobre datos de telemetría (Opcional)

Url Video de apoyo:
https://drive.google.com/open?id=1o2cz9UgBfjgyiaP5hq54z_r6HgaNF9Xw
1. Ir a la pestaña de reglas, En agregar regla y crear una nueva



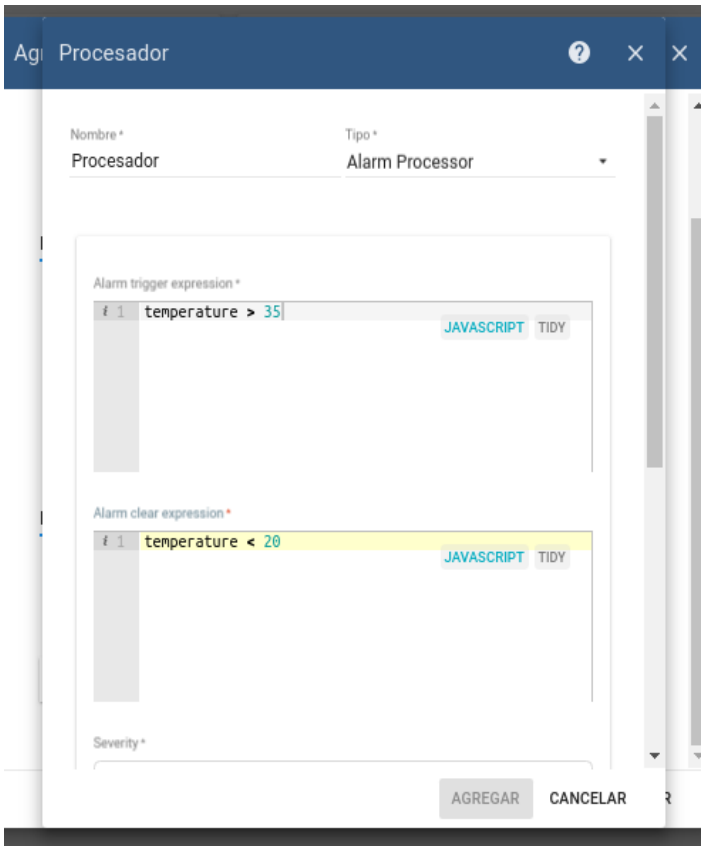
En este caso la alarma se disparará cuando un dato de temperatura sea mayor a 36.

2. Se agrega un filtro, en el botón de agregar en la parte de filtros, se configura como en la imagen y se agrega.

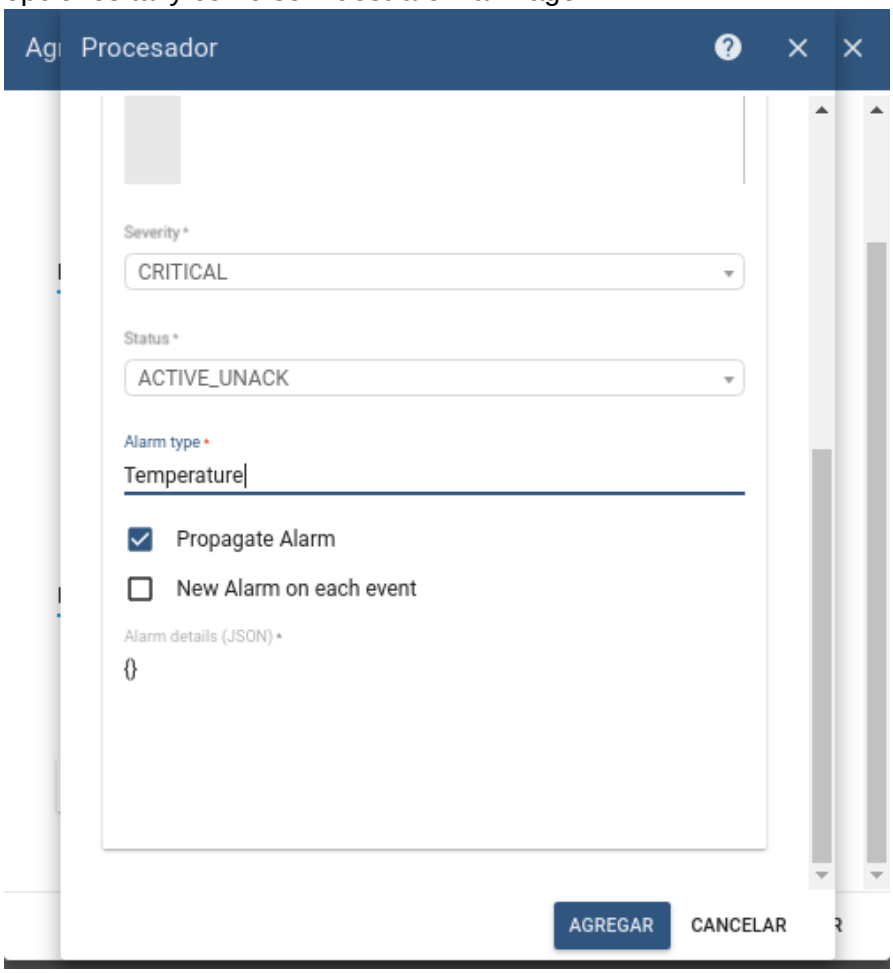


Y posteriormente otro filtro como el que se muestra en la siguiente imagen:

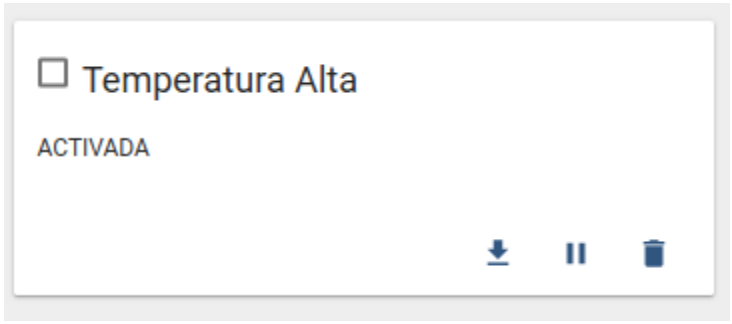
3. Se procede a crear un procesador, y se configura tal y como se muestra en la imagen, en este caso se disparará la alarma cuando la temperatura sea mayor a 35 y se borrará cuando sea menor a 20



4. Se procede a elegir la severidad de la alarma, dependiendo de la severidad escogida se mostrará un color distinto en el panel de Thingsboard, en este caso escogeremos las opciones tal y como se muestra en la imagen



5. Agregamos la alarma en el botón de agregar, y la activamos

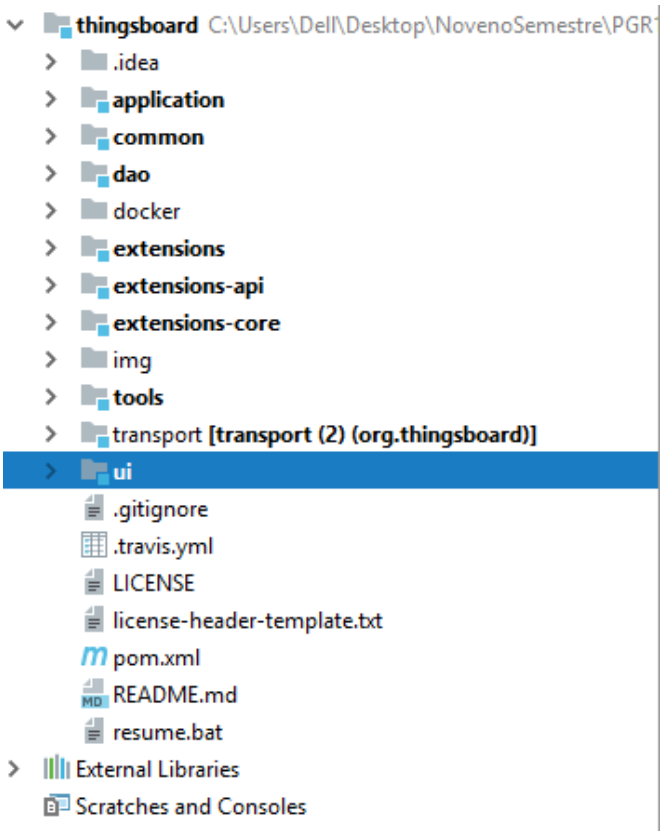


Cuando se envíen datos que cumplan la condición podrá ver en el panel asociado el cambio de color en el dispositivo.

5.4. Manual técnico

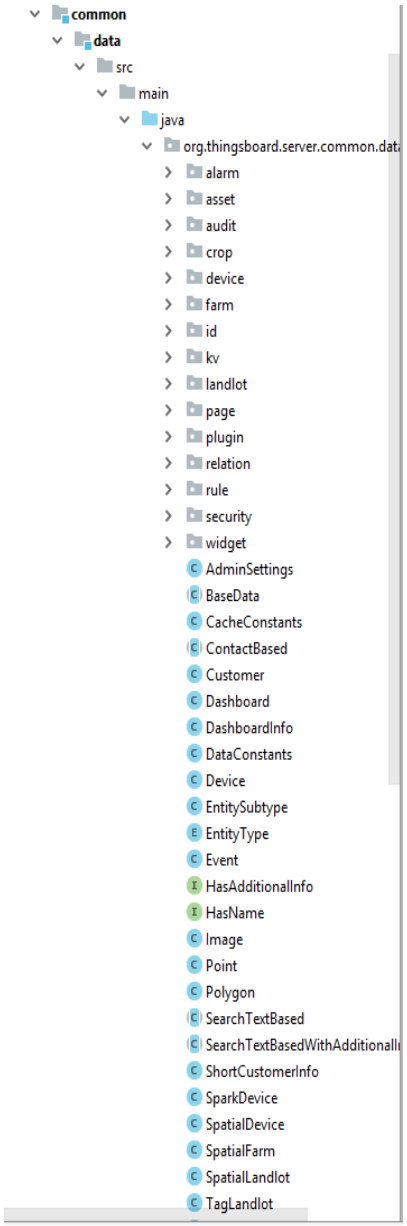
5.4.1 Back-end:

El código fuente de Thingsboard cuenta con 10 proyectos o paquetes principales, donde cada uno se encarga de un o varios aspectos funcionales para que la plataforma ejecute correctamente.

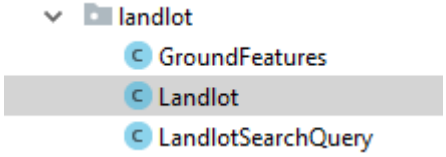


5.4.1.1 Conceptos (Clases)

En este caso como se quiere agregar un nuevo concepto (ej: finca, lote, cultivo, etc), toda la atención recaerá sobre 3 paquetes o proyectos: application, common y dao. Comenzamos por el paquete common, este se encarga de administrar las clases u objetos que maneja Thingsboard, como se aprecia en la imagen de abajo, cada objeto posee su propia carpeta, ya que el objeto o la clase por sí sola no puede funcionar correctamente, así que necesitará de otras clases u objetos para lograrlo. También se puede ver que otras clases no tienen carpeta, ya que estos son objetos utilizados por otras clases.



Para comenzar a agregar un concepto, se tomará el ejemplo del objeto Landlot.



Esta clase es fácil de entender, ya que no posee métodos complicados, solo tiene los atributos
que el usuario quiera agregar, con sus respectivos constructores, siempre toca tener el constructor vacíos o si no van a haber conflictos con la base de datos Cassandra.

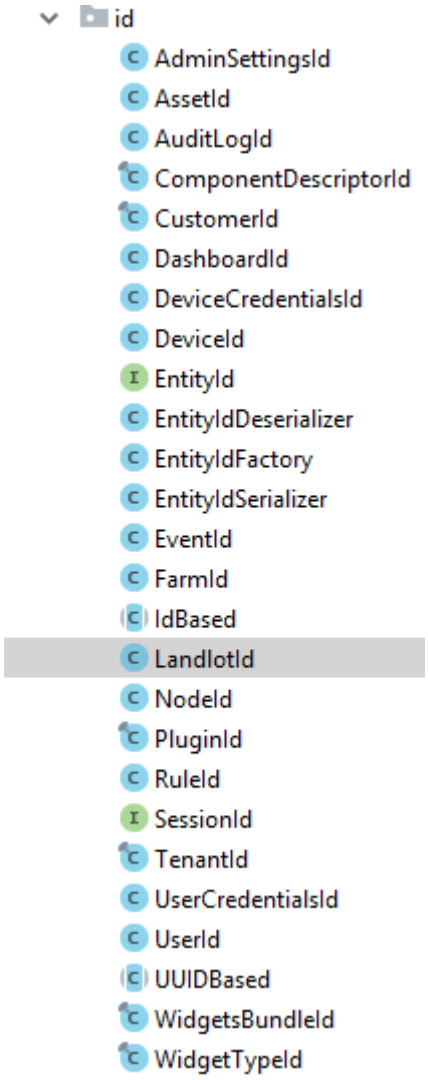
```java
@EqualsAndHashCode(callSuper = true)
public class Landlot extends SearchTextBasedWithAdditionalInfo<LandlotId> implements HasName {

    private static final long serialVersionUID = 2807343040519543363L;

    private TenantId tenantId;
    private CustomerId customerId;
    private String name;
    private String type;
    private String farmId;
    private Polygon location;
    private Crop crop;
    private List<Crop> cropsHistory;
    private Area totalArea;
    private GroundFeatures groundFeatures;
    private List<UUID> devices;


    public Landlot() { super(); }

    public Landlot(LandlotId id) { super(id); }

    public Landlot(Landlot landlot){
        super(landlot);
        this.tenantId = landlot.getTenantId();
        this.customerId = landlot.getCustomerId();
        this.name = landlot.getName();
        this.type = landlot.getType();
        this.farmId = landlot.getFarmId();
        this.setCrop(landlot.getCrop());
        this.cropsHistory = landlot.getCropsHistory();
        this.totalArea = landlot.getTotalArea();
        this.groundFeatures = landlot.getGroundFeatures();
        this.devices=landlot.getDevices();
    }
```

También toca agregar los get y set de cada atributo, principalmente para que sea más sencillo guardar el objeto en Cassandra y obviamente para que el usuario pueda utilizar el objeto más fácilmente.

```
public void setLocation(Polygon location) { this.location = location; }

public Crop getCrop() { return crop; }

public void setCrop(Crop crop) { this.crop = crop; }

public List<Crop> getCropsHistory() { return cropsHistory; }

public void setCropsHistory(List<Crop> cropsHistory) { this.cropsHistory = cropsHistory; }

public Area getTotalArea() { return totalArea; }

public void setTotalArea(Area totalArea) { this.totalArea = totalArea; }

public GroundFeatures getGroundFeatures() { return groundFeatures; }

public void setGroundFeatures(GroundFeatures groundFeatures) { this.groundFeatures = groundFeatures; }

public List<UUID> getDevices() { return devices; }

public void setDevices(List<UUID> devices) { this.devices = devices; }
```

Como se mostró al principio el paquete Landlot contiene otras dos clases, ya que la clase Landlot, las utiliza, pero esto queda a la libertad del usuario.

Dentro de la carpeta id, se encuentran todas las clases que representan un identificador único de cada objeto o clase del proyecto common.

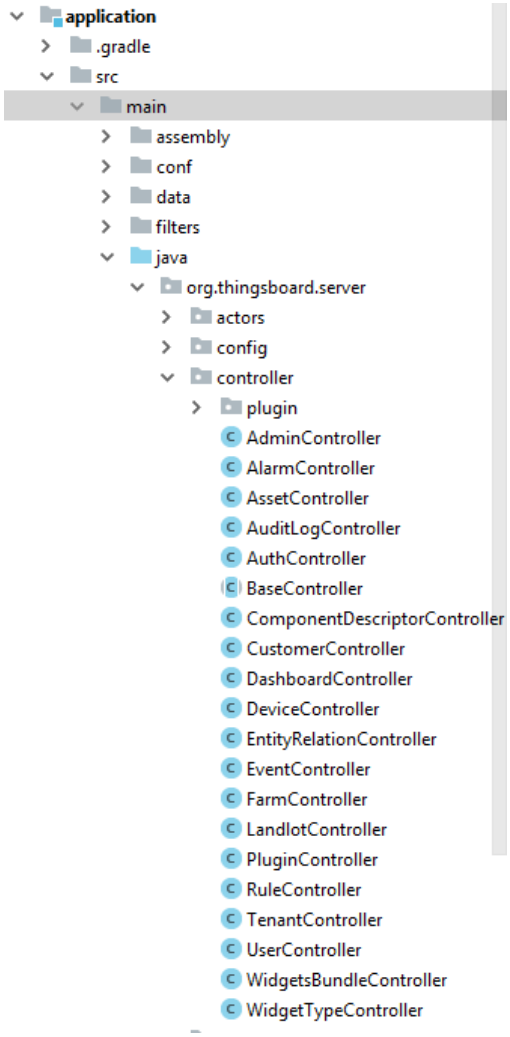


Observando la clase LandlotId, esta utiliza una clase llamada UUID la cual se basa en las fechas de creación de los objetos para poder crear los identificadores únicos que representan
a cada objeto. Cada objeto tiene su propio EntityType para ser reconocido más fácilmente.

```java
public class LandlotId extends UUIDBased implements EntityId {

    private static final long serialVersionUID = 1L;

    @JsonCreator
    public LandlotId(@JsonProperty("id") UUID id) { super(id); }

    public static LandlotId fromString(String landlotId) { return new LandlotId(UUID.fromString(landlotId)); }

    @JsonIgnore
    @Override
    public EntityType getEntityType() { return EntityType.LANDLOT; }
}
```

5.4.1.2 Controladores

Seguimos con el proyecto **application**, quien contiene los controladores de cada objeto y estos controladores utilizan los servicios que ofrece el proyecto dao que se explicará más adelante.



Comenzando por los controladores tomaremos el ejemplo de LandlotController, como se aprecia en la imagen de abajo, todos los controladores extienden a la clase BaseController ya que esta contiene todos los servicios para que lo utilicen los controladores. Se utiliza el API
REST que se base en el protocolo HTTP para realizar sus funciones y hacer comunicaciones con el front-end de la aplicación.

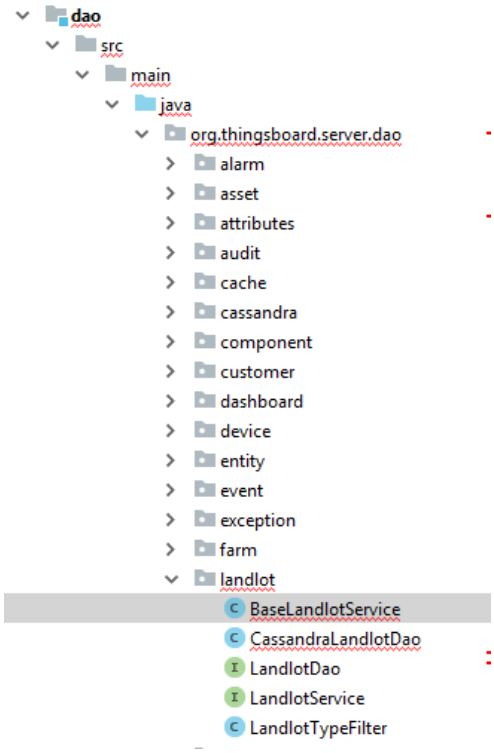
```
@RestController
@RequestMapping("/api")
public class LandlotController extends BaseController {
```

Para entender mejor esta clase tomaremos uno de sus métodos llamado saveLandlot; cómo la plataforma Thingsboard cuenta con seguridad y roles, cada uno de estos métodos cuentan con una anotación “@PreAthorize” lo que significa que esos roles que tienen en este caso **'TENANT_ADMIN'** y **'CUSTOMER_USER'** son los únicos que pueden hacer uso de este método, luego sigue todo lo normal que tiene un método en un controlador REST, cómo el recurso donde se guarda el objeto, en este caso “/landlot”, el método HTTP que se utiliza,cómo se está guardando un objeto Landlot se usa POST y por último el cuerpo del método que el objeto debe cumplir ciertas condiciones para poderse guardar, tales como cumplir los requisitos de seguridad anteriormente mencionados, verificar si el objeto ya existe en la base de datos, y si ya existe entonces lo actualiza y por último el manejo de excepciones.

```
@PreAuthorize("hasAnyAuthority('TENANT_ADMIN', 'CUSTOMER_USER')")
@RequestMapping(value = "/landlot", method = RequestMethod.POST)
@ResponseBody
public Landlot saveLandlot(@RequestBody Landlot landlot) throws ThingsboardException {
    try {
        landlot.setTenantId(getCurrentUser().getTenantId());
        if (getCurrentUser().getAuthority() == Authority.CUSTOMER_USER) {
            if (landlot.getId() == null || landlot.getId().isNullUid()
                    || landlot.getCustomerId() == null || landlot.getCustomerId().isNullUid()) {
                throw new ThingsboardException("You don't have permission to perform this operation!",
                        ThingsboardErrorCode.PERMISSION_DENIED);
            } else {
                checkCustomerId(landlot.getCustomerId());
            }
        }
        Landlot savedLandlot = checkNotNull(landlotService.saveLandlot(landlot));
        SpatialLandlot spatialLandlot = new SpatialLandlot(savedLandlot.getId().getId().toString(), savedLandlot.getFarmId(), landlot.getLocation());
        if (landlot.getLocation() != null) {
            if (mongoService.getMongodbFarm().checkCropInFarm(landlot.getLocation(), landlot.getFarmId())) {
                mongoService.getMongodblandlot().save(spatialLandlot);
            } else {
                throw new IncorrectParameterException("Polygon not contains in farm!");
            }
        } else {
            mongoService.getMongodblandlot().save(spatialLandlot);
        }
        return savedLandlot;

    } catch (Exception e) {
        logEntityAction(emptyId(EntityType.LANDLOT), landlot,
                customerId: null, landlot.getId() == null ? ActionType.ADDED : ActionType.UPDATED, e);
        throw handleException(e);
    }
}
```

5.4.1.3 Servicios y Base de datos (Cassandra)

Los servicios son interfaces, estos son implementados por clases del proyecto **dao**, ya que son los que interactúan con la base de datos, en la interfaz LandlotService, se pueden ver los métodos que se utilizan.



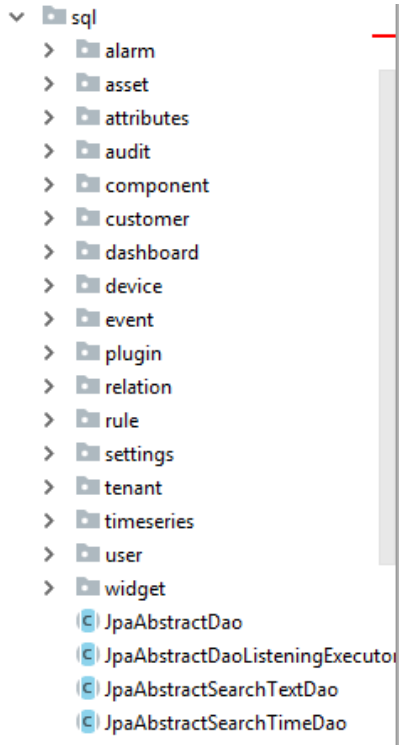

```java
public interface LandlotService {

    Landlot findLandlotById(LandlotId landlotId);

    Map<String, TreeMap<Long,Double>> getHistoricalValues(String landlotId, long minDate, long maxDate);

    ListenableFuture<Landlot> findLandlotByIdAsync(LandlotId landlotId);

    Optional<Landlot> findLandlotByTenantIdAndName(TenantId tenantId, String name);

    Landlot saveLandlot(Landlot landlot);

    Landlot assignLandlotToCustomer(LandlotId landlotId, CustomerId customerId);

    Landlot unassignLandlotFromCustomer(LandlotId landlotId);

    void deleteLandlot(LandlotId landlotId);

    TextPageData<Landlot> findLandlotsByTenantId(TenantId tenantId, TextPageLink pageLink);

    TextPageData<Landlot> findLandlotsByTenantIdAndType(TenantId tenantId, String type, TextPageLink pageLink);

    ListenableFuture<List<Landlot>> findLandlotsByTenantIdAndIdsAsync(TenantId tenantId, List<LandlotId> landlotIds);

    void deleteLandlotsByTenantId(TenantId tenantId);

    TextPageData<Landlot> findLandlotsByTenantIdAndCustomerId(TenantId tenantId, CustomerId customerId, TextPageLink pageLink);

    TextPageData<Landlot> findLandlotsByTenantIdAndCustomerIdAndType(TenantId tenantId, CustomerId customerId, String type, TextPageLink pageLink);

    ListenableFuture<List<Landlot>> findLandlotsByTenantIdCustomerIdAndIdsAsync(TenantId tenantId, CustomerId customerId, List<LandlotId> landlotIds);

    void unassignCustomerLandlots(TenantId tenantId, CustomerId customerId);

    ListenableFuture<List<Landlot>> findLandlotsByQuery(LandlotSearchQuery query);

    ListenableFuture<List<EntitySubtype>> findLandlotTypesByTenantId(TenantId tenantId);

    ListenableFuture<List<LandlotEntity>> findLandlotsByFarmId(String farmId);

    ListenableFuture<List<LandlotEntity>> allLandlots();
}
```

Thingsboard, para la persistencia de objetos y datos, utiliza una base de datos no relacional o lo que sería lo mismo una base de datos NOSQL, como lo es Cassandra, por lo que las clases que implementa los servicios anteriormente mencionados, primero utilizan un objeto llamado Base(Objeto) Service, o en el caso del objeto Landlot, BaseLandlotService, que tiene un objeto llamado CassandraLandlotDao, quien es el responsable de hacer las consultas a la base de datos Cassandra. Solo se muestra una parte del código ya que estas clases tienen muchos métodos.

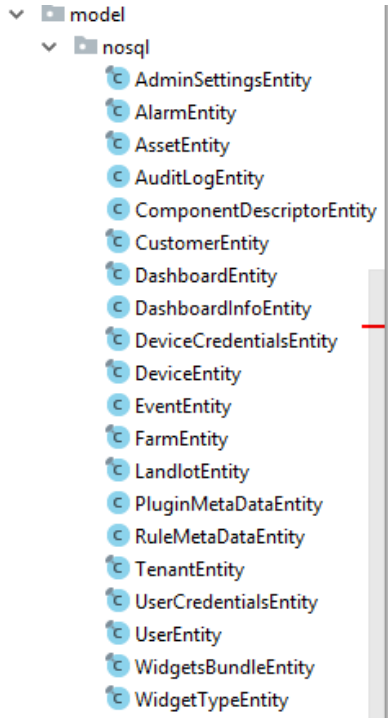
```
@Service
@Slf4j
public class BaseLandlotService extends AbstractEntityService implements LandlotService {

    public static final String INCORRECT_TENANT_ID = "Incorrect tenantId ";
    public static final String INCORRECT_PAGE_LINK = "Incorrect page link ";
    public static final String INCORRECT_CUSTOMER_ID = "Incorrect customerId ";
    public static final String INCORRECT_LANDLOT_ID = "Incorrect landlotId ";
    @Autowired
    private LandlotDao landlotDao;

    @Autowired
    private AttributesDao attributesDao;

    @Autowired
    private TenantDao tenantDao;

    @Autowired
    private CustomerDao customerDao;

    @Override
    public Landlot findLandlotById(LandlotId landlotId) {
        log.trace("Executing findLandlotById [{}]", landlotId);
        validateId(landlotId,  errorMessage: INCORRECT_LANDLOT_ID + landlotId);
        return landlotDao.findById(landlotId.getId());
    }

    private List<Date> getDaysBetweenDates(Date startDate, Date endDate){
        List<Date> dates = new ArrayList<>();
        Calendar calendar = new GregorianCalendar();
        calendar.setTime(startDate);
        while(calendar.getTime().before(endDate)){
            Date result = calendar.getTime();
            dates.add(result);
            calendar.add(Calendar.DATE,  amount: 1);
        }
        return dates;
    }
```

```java
@Component
@Slf4j
@SqlDao
public class CassandraLandlotDao extends CassandraAbstractSearchTextDao<LandlotEntity, Landlot> implements LandlotDao {

    @Override
    protected Class<LandlotEntity> getColumnFamilyClass() { return LandlotEntity.class; }

    @Override
    protected String getColumnFamilyName() { return LANDLOT_COLUMN_FAMILY_NAME; }

    @Override
    public Landlot save(Landlot domain) {
        Landlot savedLandlot = super.save(domain);
        EntitySubtype entitySubtype = new EntitySubtype(savedLandlot.getTenantId(), EntityType.LANDLOT, savedLandlot.getType());
        EntitySubtypeEntity entitySubtypeEntity = new EntitySubtypeEntity(entitySubtype);
        Statement saveStatement = cluster.getMapper(EntitySubtypeEntity.class).saveQuery(entitySubtypeEntity);
        executeWrite(saveStatement);
        return savedLandlot;
    }

    @Override
    public List<Landlot> findLandlotsByTenantId(UUID tenantId, TextPageLink pageLink) {
        log.debug("Try to find landlots by tenantId [{}] and pageLink [{}]", tenantId, pageLink);
        List<LandlotEntity> landlotEntities = findPageWithTextSearch(LANDLOT_BY_TENANT_AND_SEARCH_TEXT_COLUMN_FAMILY_NAME,
                Collections.singletonList(eq(LANDLOT_TENANT_ID_PROPERTY, tenantId)), pageLink);

        log.trace("Found landlots [{}] by tenantId [{}] and pageLink [{}]", landlotEntities, tenantId, pageLink);
        return DaoUtil.convertDataList(landlotEntities);
    }
```

Anteriormente se dijo que Thingsboard trabaja con bases de datos no relacionales, pero también da la posibilidad de trabajar con bases de datos SQL, esto depende del gusto del programador, para poder cambiar el tipo de base de datos toca hacer una configuración a un archivo llamado “thingsboard.yml”, esto se explica en el manual de instalación.



Para terminar el proyecto dao también tiene clases importantes para que se puedan guardar los objetos en Cassandra de manera mucho más sencilla, estas clases se llaman (Objeto)Entity, en el caso del Landlot, LandlotEntity.



Estas clases son unas de las más cercanas a la base de datos ya que son las que configuran los objetos para guardarse en las tablas de Cassandra. Como se puede observar en la imagen de abajo cada atributo del objeto se va a guardar en una columna, y se manejan de tipo String para cualquier tipo de objeto y de tipo UUID para los identificadores. En las anotaciones “@Column” aparecen unos nombres en mayúscula que representan los nombres de las columnas de la tabla en cuestión, estos nombres o Strings los contiene otra clase llamada ModelConstans, que se explicará más adelante.

```java
@ToString
public final class LandlotEntity implements SearchTextEntity<Landlot> {


    public String getFarmId() { return farmId; }

    public void setFarmId(String farmId) { this.farmId = farmId; }

    @PartitionKey(value = 0)
    @Column(name = ID_PROPERTY)
    private UUID id;

    @PartitionKey(value = 1)
    @Column(name = LANDLOT_TENANT_ID_PROPERTY)
    private UUID tenantId;

    @PartitionKey(value = 2)
    @Column(name = LANDLOT_CUSTOMER_ID_PROPERTY)
    private UUID customerId;

    @PartitionKey(value = 3)
    @Column(name = LANDLOT_TYPE_PROPERTY)
    private String type;

    @Column(name = LANDLOT_NAME_PROPERTY)
    private String name;

    @Column(name = LANDLOT_FARMID_PROPERTY)
    private String farmId;

    @Column(name = SEARCH_TEXT_PROPERTY)
    private String searchText;

    @com.datastax.driver.mapping.annotations.Column(name = LANDLOT_ADDITIONAL_INFO_PROPERTY, codec = JsonCodec.class)
    private JsonNode additionalInfo;

    @Column(name = LANDLOT_CROP)
    private String crop;

    @Column(name = LANDLOT_CROPS_HISTORY)
    private String cropsHistory;

    @Column(name = LANDLOT_TOTAL_AREA)
    private String totalArea;

    @Column(name = GROUND_FEATURES)
    private String groundFeatures;

    @Column(name = LANDLOT_DEVICES)
    private String devices;
```

Esta clase consta de dos métodos importantes, el primero se llama LandlotEntity, en este caso es el constructor, el cual obtiene todos los atributos del objeto por medio de los get y los sets para convertirlos a formato JSON, ya que Cassandra trabaja de manera más sencilla con tipos simples.

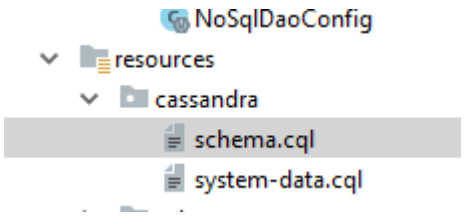
```java
public LandlotEntity() { super(); }

public LandlotEntity(Landlot landlot) {
    if (landlot.getId() != null) {
        this.id = landlot.getId().getId();
    }
    if (landlot.getTenantId() != null) {
        this.tenantId = landlot.getTenantId().getId();
    }
    if (landlot.getCustomerId() != null) {
        this.customerId = landlot.getCustomerId().getId();
    }
    this.name = landlot.getName();
    this.type = landlot.getType();
    this.farmId = landlot.getFarmId();
    this.additionalInfo = landlot.getAdditionalInfo();
    try {
        ObjectMapper mapper = new ObjectMapper();
        this.crop = mapper.writeValueAsString(landlot.getCrop());
        this.cropsHistory = mapper.writeValueAsString(landlot.getCropsHistory());
        this.totalArea = mapper.writeValueAsString(landlot.getTotalArea());
        this.groundFeatures = mapper.writeValueAsString(landlot.getGroundFeatures());
        this.setDevices(mapper.writeValueAsString(landlot.getDevices()));
    } catch (JsonProcessingException e) {
        e.printStackTrace();
    }
}
```

El segundo método se llama toData, este se encarga de cuando se saca el objeto de Cassandra, convierte los atributos en formato JSON a su tipo de objeto original para luego formar el objeto Landlot, este método se utiliza cuando se quiere obtener un objeto de la base de datos.

```java
@Override
public Landlot toData() {
    Landlot landlot = new Landlot(new LandlotId(id));
    landlot.setCreatedTime(UUIDs.unixTimestamp(id));
    if (tenantId != null) {
        landlot.setTenantId(new TenantId(tenantId));
    }
    if (customerId != null) {
        landlot.setCustomerId(new CustomerId(customerId));
    }
    landlot.setName(name);
    landlot.setType(getType());
    landlot.setFarmId(farmId);
    landlot.setAdditionalInfo(additionalInfo);
    try {
        ObjectMapper mapper = new ObjectMapper();
        landlot.setCrop(mapper.readValue(crop, Crop.class));
        landlot.setCropsHistory(mapper.readValue(cropsHistory, mapper.getTypeFactory().constructParametricType(List.class, Crop.class)));
        landlot.setTotalArea(mapper.readValue(totalArea, Area.class));
        landlot.setGroundFeatures(mapper.readValue(groundFeatures, GroundFeatures.class));
        landlot.setDevices(mapper.readValue(getDevices(), mapper.getTypeFactory().constructParametricType(List.class, UUID.class)));
    } catch (IOException e) {
        e.printStackTrace();
    }

    return landlot;
}
```

La clase ModelConstans, anteriormente mencionada, contiene atributos que son de tipo String, estáticos y finales, que como ya se había dicho, representan a las columnas de una tabla en especifico, esta clase tiene una relación con un archivo de configuración, quién es el encargado de crear las tablas y las columnas en Cassandra, este archivo se llama schema.cql.

```
/**
 * Cassandra landlot constants.
 */
public static final String LANDLOT_COLUMN_FAMILY_NAME = "landlot";
public static final String LANDLOT_TENANT_ID_PROPERTY = TENANT_ID_PROPERTY;
public static final String LANDLOT_CUSTOMER_ID_PROPERTY = CUSTOMER_ID_PROPERTY;
public static final String LANDLOT_NAME_PROPERTY = "name";
public static final String LANDLOT_TYPE_PROPERTY = "type";
public static final String LANDLOT_FARMID_PROPERTY = "farm_id";
public static final String LANDLOT_CROP = "crop";
public static final String LANDLOT_CROPS_HISTORY = "crops_history";
public static final String LANDLOT_TOTAL_AREA = "total_area";
public static final String GROUND_FEATURES = "ground_features";
public static final String LANDLOT_DEVICES = "devices";
public static final String LANDLOT_ADDITIONAL_INFO_PROPERTY = ADDITIONAL_INFO_PROPERTY;

public static final String LANDLOT_BY_TENANT_AND_SEARCH_TEXT_COLUMN_FAMILY_NAME = "landlot_by_tenant_and_search_text";
public static final String LANDLOT_BY_TENANT_BY_TYPE_AND_SEARCH_TEXT_COLUMN_FAMILY_NAME = "landlot_by_tenant_by_type_and_search_text";
public static final String LANDLOT_BY_CUSTOMER_AND_SEARCH_TEXT_COLUMN_FAMILY_NAME = "landlot_by_customer_and_search_text";
public static final String LANDLOT_BY_CUSTOMER_BY_TYPE_AND_SEARCH_TEXT_COLUMN_FAMILY_NAME = "landlot_by_customer_by_type_and_search_text";
public static final String LANDLOT_BY_TENANT_AND_NAME_VIEW_NAME = "landlot_by_tenant_and_name";
public static final String LANDLOT_TYPES_BY_TENANT_VIEW_NAME = "landlot_types_by_tenant";
```



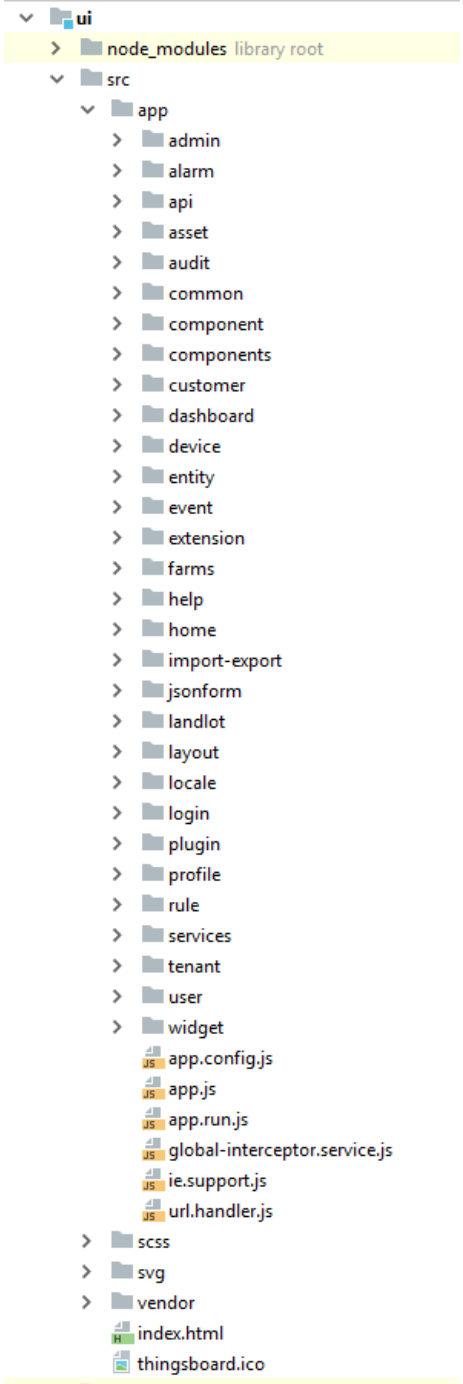
Como se puede observar, este archivo contiene los comandos para crear la tabla con sus columnas y si el desarrollador lo ve necesario puede también crear vistas de la tabla, lo que toca tener en cuenta es que los nombres de las columnas tienen que ser iguales en la clase ModelConstans, que están siendo representadas por los Strings estáticos y finales para que funcione correctamente. Si se quiere agregar otro concepto se crea la tabla en este archivo de configuración con sus respectivas columnas que representan los atributos de dicho concepto.

```
CREATE TABLE IF NOT EXISTS thingsboard.landlot (
    id timeuuid,
    tenant_id timeuuid,
    customer_id timeuuid,
    name text,
    type text,
    farm_id text,
    search_text text,
    additional_info text,
    crop text,
    crops_history text,
    total_area text,
    ground_features text,
    devices text,
    PRIMARY KEY (id, tenant_id, customer_id, type)
);

CREATE MATERIALIZED VIEW IF NOT EXISTS thingsboard.all_landlots AS
    SELECT *
    from thingsboard.landlot
    WHERE tenant_id IS NOT NULL AND customer_id IS NOT NULL AND type IS NOT NULL AND name IS NOT NULL AND id IS NOT NULL
    PRIMARY KEY (id, tenant_id, customer_id, type);

CREATE MATERIALIZED VIEW IF NOT EXISTS thingsboard.landlot_by_tenant_and_name AS
    SELECT *
    from thingsboard.landlot
    WHERE tenant_id IS NOT NULL AND customer_id IS NOT NULL AND type IS NOT NULL AND name IS NOT NULL AND id IS NOT NULL
    PRIMARY KEY ( tenant_id, name, id, customer_id, type)
    WITH CLUSTERING ORDER BY ( name ASC, id DESC, customer_id DESC);

CREATE MATERIALIZED VIEW IF NOT EXISTS thingsboard.landlot_by_tenant_and_search_text AS
    SELECT *
    from thingsboard.landlot
    WHERE tenant_id IS NOT NULL AND customer_id IS NOT NULL AND type IS NOT NULL AND search_text IS NOT NULL AND id IS NOT NULL
    PRIMARY KEY ( tenant_id, search_text, id, customer_id, type)
    WITH CLUSTERING ORDER BY ( search_text ASC, id DESC, customer_id DESC);

CREATE MATERIALIZED VIEW IF NOT EXISTS thingsboard.landlot_by_tenant_by_type_and_search_text AS
    SELECT *
    from thingsboard.landlot
    WHERE tenant_id IS NOT NULL AND customer_id IS NOT NULL AND type IS NOT NULL AND search_text IS NOT NULL AND id IS NOT NULL
    PRIMARY KEY ( tenant_id, type, search_text, id, customer_id)
    WITH CLUSTERING ORDER BY ( type ASC, search_text ASC, id DESC, customer_id DESC);

CREATE MATERIALIZED VIEW IF NOT EXISTS thingsboard.landlot_by_customer_and_search_text AS
    SELECT *
    from thingsboard.landlot
    WHERE tenant_id IS NOT NULL AND customer_id IS NOT NULL AND type IS NOT NULL AND search_text IS NOT NULL AND id IS NOT NULL
    PRIMARY KEY ( customer_id, tenant_id, search_text, id, type )
    WITH CLUSTERING ORDER BY ( tenant_id DESC, search_text ASC, id DESC );
```

5.4.2 Front-end:

La parte gráfica de Thingsboard se desarrolla en el proyecto **ui**, donde cada concepto tiene su propia carpeta. Esta sección está desarrollada en html5, javascript y ccs, en otras palabras, está hecho en Angularjs.

5.4.2.1 Archivos principales



Este proyecto tiene dos archivos principales en donde se controla toda la parte gráfica, el index.html, el cual contiene todos los archivos .html del proyecto ui, en donde se pueden agregar scripts, para tener nuevos estilos.

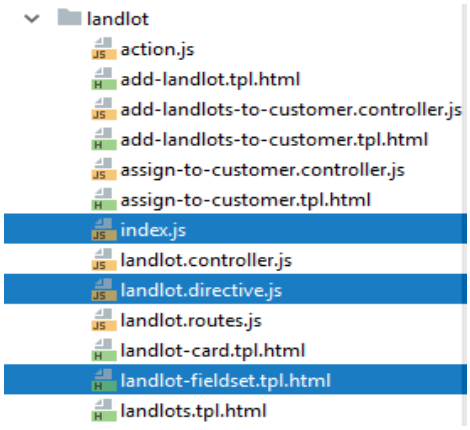
```
-->
<!DOCTYPE html>
<html ng-app="thingsboard" ng-strict-di style="...">
  <head>
    <title ng-bind="pageTitle"></title>
    <base href="/" />
    <meta charset="utf-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="description" content="" />
    <meta name="viewport" content="initial-scale=1, maximum-scale=1, user-scalable=no" />
    <link rel="icon"
          type="image/x-icon"
          href="static/thingsboard.ico" />
    <script async defer src="https://maps.googleapis.com/maps/api/js?key=AIzaSyCmzk5bzRpwDOvjmDGea2hcCgtO3fPr3VM" type="text/javascript"></script>
    <script src="http://code.highcharts.com/stock/highstock.src.js"></script>
    <style type="text/css">
        .chart .highcharts-tooltip > span{ width: 175px;float: left; margin: 0 }
        .chart .highcharts-tooltip .header{ font-weight: bold; width: 154px!important; height: auto!important; float: left; border-bottom: 1px solid #bebebe;
            -webkit-border-top-left-radius: 5px;
            -webkit-border-top-right-radius: 5px;
            -moz-border-radius-topleft: 5px;
            -moz-border-radius-topright: 5px;
            border-top-left-radius: 5px;
            border-top-right-radius: 5px;}
        .chart .highcharts-tooltip .circle{width: 12px; height: 12px; border-radius: 24px; float: left;  margin: 2px 5px 5px 10px;clear: left}
        .chart .highcharts-tooltip p.country{float: left; margin-right: 0}
        .chart .highcharts-tooltip p{float: right; font-size: 12px; margin-right: 10px;}
    </style>
  </head>
  <body>
    <ui-view layout="row" layout-fill>
    </ui-view>
  </body>
</html>
```

También está el archivo app.js, quien controla todos los módulos javascript que controla los archivos .html. Cada concepto u objeto, como por ejemplo Landlot, tiene un archivo llamado index.js con el cual controla todo lo que es suyo, ese index.js se tiene que inyectar en el app.js para que la plataforma lo reconozca.

```
angular.module('thingsboard', [
    ngMaterial,
    ngMdIcons,
    ngCookies,
    angularSocialshare,
    'pascalprecht.translate',
    'mdColorPicker',
    mdPickers,
    ngSanitize,
    vAccordion,
    ngAnimate,
    'ngWebSocket',
    angularJwt,
    'dndLists',
    mdDataTable,
    'material.components.expansionPanels',
    ngTouch,
    'angular-carousel',
    'ngclipboard',
    react.name,
    'flow',
    thingsboardLocales,
    thingsboardLogin,
    thingsboardDialogs,
    thingsboardMenu,
    thingsboardRaf,
    thingsboardUtils,
    thingsboardDashboardUtils,
    thingsboardTypes,
    thingsboardApiTime,
    thingsboardKeyboardShortcut,
    thingsboardHelp,
    thingsboardToast,
    thingsboardClipboard,
    thingsboardHome,
    thingsboardApiLogin,
    thingsboardApiDevice,
    thingsboardApiUser,
    thingsboardApiEntityRelation,
    thingsboardApiAsset,
    thingsboardApiFarm,
    thingsboardApiLandlot,
    thingsboardApiAttribute,
    thingsboardApiEntity,
    thingsboardApiAlarm,
    thingsboardApiAuditLog,
    uiRouter])
    .config(AppConfig)
    .factory('globalInterceptor', GlobalInterceptor)
    .run(AppRun);
```

5.4.2.2 Archivos de configuración por concepto o clase

Tomando otra vez el ejemplo de Landlot, como se puede apreciar en la imagen de abajo, cuenta con varios archivos, algunos html y otros javascritp, pero en este caso, nos centraremos en tres archivos específicos: landlot.directive.js, index.js y landlot-fieldset.tpl.html.



Empezando por el **index.js**, como se dijo anteriormente, es el que controla todos los archivos de la carpeta “landlot”, se definen cuales archivos son controladores, directivas y configuraciones, también se pueden agregar otras dependencias para agregar otros estilos.

```
import uiRouter from 'angular-ui-router';
import 'highcharts-ng/dist/highcharts-ng.js';
import thingsboardGrid from '../components/grid.directive';
import thingsboardApiUser from '../api/user.service';
import thingsboardApiLandlot from '../api/landlot.service';
import thingsboardApiCustomer from '../api/customer.service';

import LandlotRoutes from './landlot.routes';
import {LandlotController, LandlotCardController} from './landlot.controller';
import AssignLandlotToCustomerController from './assign-to-customer.controller';
import AddLandlotsToCustomerController from './add-landlots-to-customer.controller';
import LandlotDirective from './landlot.directive';

export default angular.module('thingsboard.landlot', [
    "highcharts-ng",
    uiRouter,
    thingsboardGrid,
    thingsboardApiUser,
    thingsboardApiLandlot,
    thingsboardApiCustomer
])
    .config(LandlotRoutes)
    .controller('LandlotController', LandlotController)
    .controller('LandlotCardController', LandlotCardController)
    .controller('AssignLandlotToCustomerController', AssignLandlotToCustomerController)
    .controller('AddLandlotsToCustomerController', AddLandlotsToCustomerController)
    .directive('tbLandlot', LandlotDirective)
    .name;
```

El **landlot.directive.js** es el archivo que controla el html o en otras palabras el que controla la parte gráfica del Landlot, pero para poder hacerlo, se necesita importar el archivo .html que va a controlar, en este caso el archivo landlot-fieldset.tpl.html, también necesita de otros componentes para poder realizar acciones, tales como, el landlotService (el servicio en la parte gráfica que se explicará más adelante), $log que se utiliza para imprimir en la consola del buscador web que se está utilizando, entre otros que se colocan como parámetros en la función “export”.

```
import landlotFieldsetTemplate from './landlot-fieldset.tpl.html';
import Action from './action';

/* eslint-enable import/no-unresolved, import/default */
/*global google*/
/*@ngInject*/
export default function LandlotDirective($compile, $templateCache, $mdDialog, toast, $translate, types, landlotService, farmService, dashboardService, customerService, $log) {
```

Para hacer más sencillo de manejar los objetos que componen a un Landlot, se simulan los objetos como funciones y dentro de ellas se colocan los atributos, para luego poderlos usar.

```
function Crop(){
    this.name = '';
    this.why = '';
    this.cause = '';
    this.startCrop = new Date();
    this.finishCropDate = null;
    this.weekens = 0;
    this.initialConditions = '';
    this.actions = [];
    this.finish = false;
    this.state = '';
    this.practices=[];
}

function Area(){
    this.extension=0.0;
    this.symbol='';
}

function GroundFeatures(){
    this.density = '';
    this.compaction = '';
    this.inclination = '';
    this.higrologicData = '';
}

function TagLandlot(){
    this.idLandlot = '';
    this.tag = '';
    this.date = 0;
    this.timeElapsed = 0;
    this.telemetryData = {};
    this.imageName = '';
    this.cropName = '';
    this.tagPolygon = new Polygon();
}
```

Por parte de los métodos o funciones que utiliza este archivo, lo hace con la ayuda de servicios que están implementados en otras carpetas (en este caso landlotService) y cómo

estas funciones se hace asíncronamente, entonces toca hacer uso del “.then” para que se puedan ejecutar.
Para que se pueda comunicar con el archivo .html, se hace uso de la variable “scope.” todo valor que tenga esta variable por delante podrá ser reconocido por el landlot-fieldset.tpl.html
.

```
scope.maxDate = scope.finishDate.getTime();
scope.minDate = scope.startDate.getTime();
scope.updateSelectedDate = function(){
    scope.selectedDate = scope.startDate;
    scope.maxDate = scope.finishDate.getTime();
    scope.minDate = scope.startDate.getTime();
    landlotService.getFilesDates(scope.minDate,scope.maxDate).then(function(response){
        scope.fechas = response;
    });
    landlotService.getHistoricalValues(scope.landlot.id.id,scope.minDate,scope.maxDate).then(function(result){
        $log.log("Este es el response de los telimetria");
        $log.log(result);
        scope.tabs.length = 0;
        dataTelemetryUpdated.length = 0;
        angular.forEach(result,function (value,key){
            scope.tabs.push(key);
            var seriesToAdd = [];
            angular.forEach(result[key], function (value, key){
                var point = [];
                point.push(Number(key));
                point.push(value);
                seriesToAdd.push(point);
            });
            var tabHistoricalValues = new infoTab(key,seriesToAdd);
            dataTelemetryUpdated.push(tabHistoricalValues);
        });
        if(dataTelemetryUpdated.length == 1){
            $log.log(dataTelemetryUpdated[0]);
            scope.highchartsNG.series.push({
                name: 'Average per day',
                data: dataTelemetryUpdated[0]['telemetry'],
                type: 'line'
            });
            scope.highchartsNG.title['text'] = dataTelemetryUpdated[0]['titleTab'];
            scope.$apply();
        }
    });
};
```

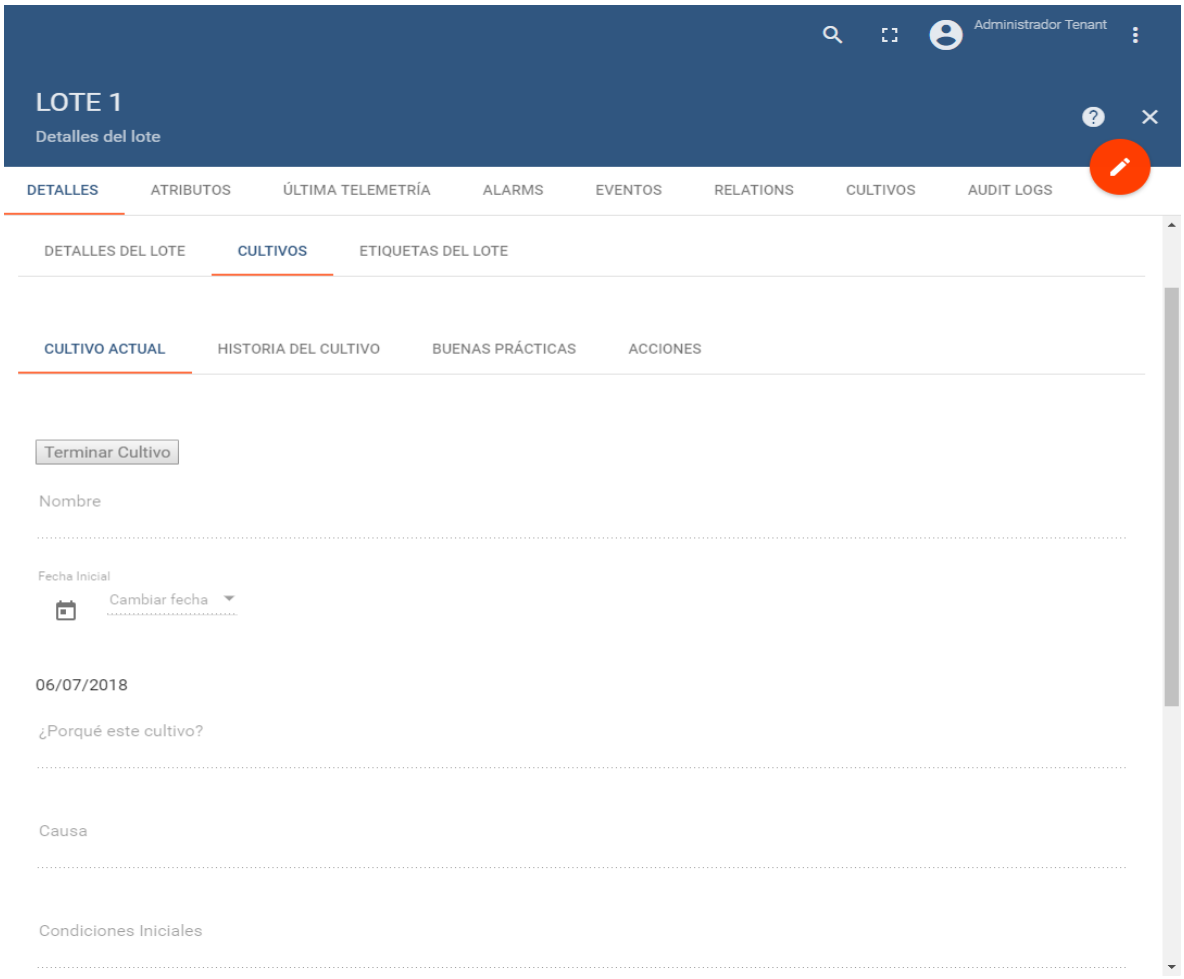
El archivo **landlot-fielset.tpl.html**, es el que se encarga mostrar por pantalla la parte gráfica del objeto Landlot, para entender mejor este archivo se mostrará un pedazo del código html y su representación gráfica en un buscador web.
Como se puede apreciar en la imagen de abajo, se empieza con una etiqueta <md-tab>, que hace referencia a una pestaña, también tiene una función llamada “ng-click” que hace referencia a una función llamada “someCrop()”.

```
scope.someCrop = function(){
    var crop = false;
    if(scope.landlot.name == null) {
        scope.landlot.crop = new Crop();
        scope.landlot.cropsHistory = [];
    }else{
        crop = true;
    }
    return crop;
};
```

Como se mencionó anteriormente todo valor que tenga la variable “scope.” por delante es reconocido por el archivo .html.
También se puede ver que la etiqueta <md-tab> tiene más etiquetas en su interior, por lo que la pestaña en la página web va a mostrar algún contenido, por ejemplo, si nos fijamos en otra etiqueta <textarea ng-model=”landlot.crop.name” rows=”2”></textarea>
Hace referencia que va a mostrar un texto (en este caso el nombre de un cultivo) obtenido de un cultivo (crop) de un lote (landlot) y lo va a mostrar en pantalla.

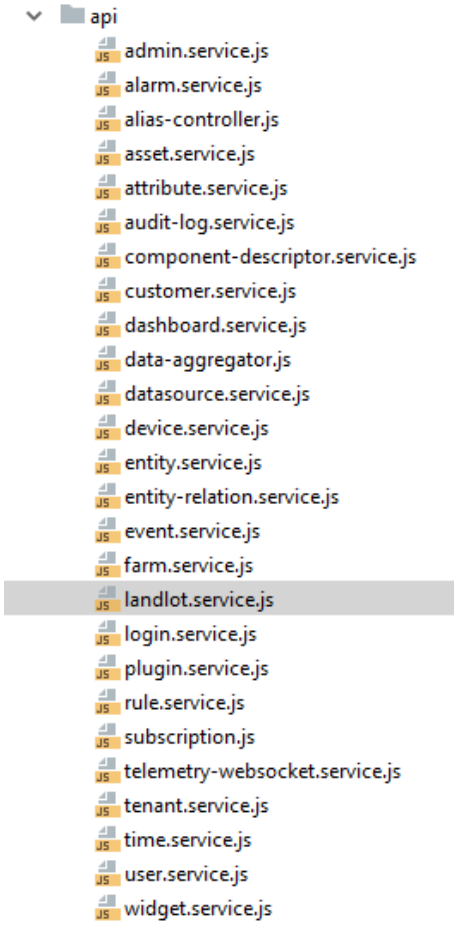
```
<md-tab label="{{ 'landlot.crops-title' | translate }}" ng-click="someCrop()">
    <md-tabs md-dynamic-height md-border-bottom>
        <md-tab label="{{ 'crop.current' | translate }}">
            <md-content class="md-padding">
                    <button ng-click="finishCrop()">{{ 'crop.finish-crop' | translate }}</button>
                    <div style="..." ng-disable="!landlot.crop.finish">
                        <md-input-container class="md-block">
                            <label translate>{{ 'crop.name' | translate }}</label>
                            <textarea ng-model="landlot.crop.name" rows="2"></textarea>
                        </md-input-container>
                        <md-input-container class="md-block">
                            <div flex-gt-xs>
                                <label>{{ 'landlot.start-date' | translate }}</label>
                                <md-datepicker ng-model="landlot.crop.startCrop"  md-placeholder="Cambiar fecha"></md-datepicker>
                            </div>
                        </md-input-container>
                        <label>{{landlot.crop.startCrop |date :  "dd/MM/y"}}</label>
                        <md-input-container class="md-block">
                            <label translate>{{ 'crop.why-crop' | translate }}</label>
                            <textarea ng-model="landlot.crop.why" rows="2"></textarea>
                        </md-input-container>
                        <md-input-container class="md-block">
                            <label translate>{{ 'crop.cause' | translate }}</label>
                            <textarea ng-model="landlot.crop.cause" rows="2"></textarea>
                        </md-input-container>
                        <md-input-container class="md-block">
                            <label translate>{{ 'crop.initial-conditions' | translate }}</label>
                            <textarea ng-model="landlot.crop.initialConditions" rows="2"></textarea>
                        </md-input-container>
                        <md-input-container class="md-block">
                            <label translate>{{ 'crop.state' | translate }}</label>
                            <textarea ng-model="landlot.crop.state" rows="2"></textarea>
                        </md-input-container>
                    </div>
                <p style="...">This is a temporal resize way for the farm tabs - Thingsboard </p>
            </md-content>
        </md-tab>
```

A continuación, se verá la representación gráfica de la fracción del código anteriormente mostrado.



5.4.2.3 Servicios y API REST

En otra carpeta llamada “api” se encuentran los ya mencionados servicios quienes son los que se comunican con el API REST del back-end. Nos vamos a centrar en landlot.service.js para poderlo explicar mejor.




```
export default angular.module('thingsboard.api.landlot', [])
    .factory('landlotService', LandlotService)
    .name;

/*@ngInject*/
function LandlotService($http, $q, customerService, userService, $log) {
```

Para realizar un método o función está también debe estar hecha en el controlador del back-end, o si no tendría sentido hacerla, por ejemplo vamos a ver el método “getAllLandlots()”, como se aprecia en la imagen de abajo, hay un aspecto que resalta el cual es el recurso “/api/Alllandlots”, el cual debe ser el mismo en el back-end para que se puedan comunicar y también se hace uso de la función “.then” ya que esto se hace de manera asíncrona.

```
function getAlllandlots(config) {
    var deferred = $q.defer();
    var landlots;
    var url = '/api/Alllandlots';
    $http.get(url,config).then(function success(response) {
        landlots=response.data;
        deferred.resolve(landlots);
    }, function fail() {
        deferred.reject();
    });
    return deferred.promise;
}
```

Método en el back-end, comparten el mismo recurso.

```
@PreAuthorize("hasAnyAuthority('TENANT_ADMIN', 'CUSTOMER_USER')")
@RequestMapping(value = "/Alllandlots", method = RequestMethod.GET)
@ResponseBody
public List<Landlot> getAllLandlots() throws ThingsboardException {
    try {
        List<LandlotEntity> landlotTypes = landlotService.allLandlots().get();
        List<Landlot> landlots = new ArrayList<>();
        for (LandlotEntity fe : landlotTypes) {
            SpatialLandlot sp = mongoService.getMongodblandlot().findById(fe.getId().toString());
            Landlot p = fe.toData();
            p.setLocation(sp.getPolygons());
            landlots.add(p);
        }
        return landlots;
    } catch (Exception e) {
        throw handleException(e);
    }
}
```

Estos métodos son utilizados por los directivos para mostrar los objetos en representaciones
gráficas en cualquier buscador web.

5.4.3 Agregar mapas en Thingsboard:

Como la documentación de Google maps lo indica toca agregar el script de google en el index.html, como se muestra en la imagen de abajo, cómo se está utilizando la versión para desarrolladores, entonces no va a tener un api key, ya conseguirlo depende del desarrollador en la página oficial de Google.

```
<!DOCTYPE html>
<html ng-app="thingsboard" ng-strict-di style="...">
  <head>
    <title ng-bind="pageTitle"></title>
    <base href="/" />
    <meta charset="utf-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="description" content="" />
    <meta name="viewport" content="initial-scale=1, maximum-scale=1, user-scalable=no" />
    <link rel="icon"
          type="image/x-icon"
          href="static/thingsboard.ico" />
    <script async defer src="https://maps.googleapis.com/maps/api/js?sensor=false" type="text/javascript"></script>
    <script src="http://code.highcharts.com/stock/highstock.src.js"></script>
    <style type="text/css">
        .chart .highcharts-tooltip > span{ width: 175px;float: left; margin: 0 }
```

Como anteriormente se dijo el directive es el que controla el html, por lo tanto el código del mapa se establece en la directiva, en este caso utilizaremos farm.directive.js, primero toca colocar /*global google*/, para la directiva lo reconozca.

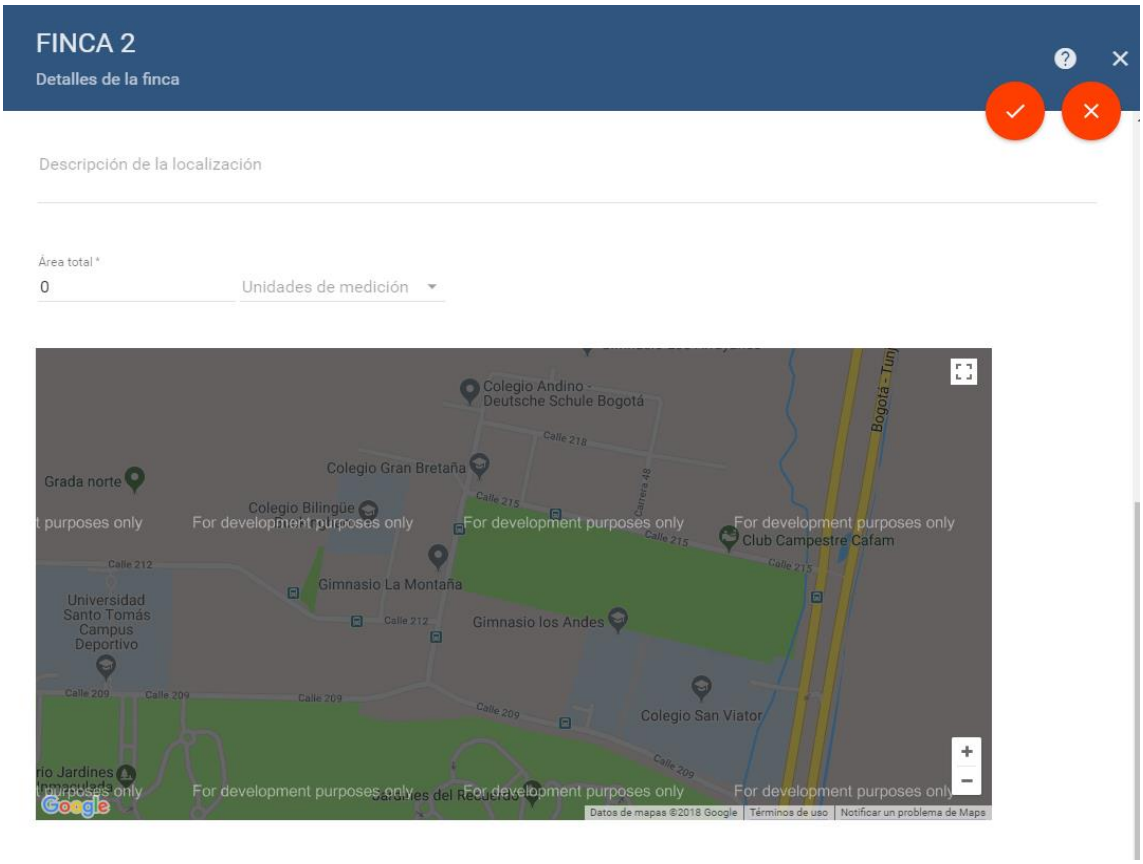
```
/* eslint-enable import/no-unresolved, import/default */
/* global google */
/* global require */
/*@ngInject*/
export default function FarmDirective($compile, $templateCache, toast, $translate, types, farmService, customerService,$log,$window) {
```

Luego pasando al código toca tener una variable “map” o cómo el desarrollador la quiera llamar a la cual se le va asignar el objeto mapa. La función se inica como normalmente se haría para inicializar un mapa de Google, pero como el estándar que se utiliza en el Angularjs de Thingsboard no es el común, toca hacer uso de la variable angular.element(#mapa)[0] para colocar el mapa en el html, el “#mapa” significa que hay una etiqueta con un id = mapa, y en la posición 0 porque el angular.element retorna un arreglo de por lo que toca escoger el primero que encuentre.

```
var map;

scope.$watch("farm",function(newVal) {
    if(newVal){
        direction();
        map = new google.maps.Map(angular.element('#mapa')[0], {
            center: {lat: scope.tempLatitude, lng: scope.tempLongitude},
            zoom: 15
        });
        dibujar();
    }
});
```

Por último, el desarrollador puede colocar el tamaño que quiera para poder visualizar el mapa.

```
<div style="min-height: 500px;">
    <div id="mapa"  style="height: 450px; width: 90%;"></div>
</div>
```



Cómo se dijo anteriormente en mapa aparece más oscuro y con letreros de “For development purpose only” que significa “solo para propósitos de desarrollo”, si el desarrollador no quiere trabajar así, deberá conseguir el api key de google maps por sus propios métodos.

5.4.4 Modelo de clasificación Árbol de decisión de la librería “mllib” de Spark:

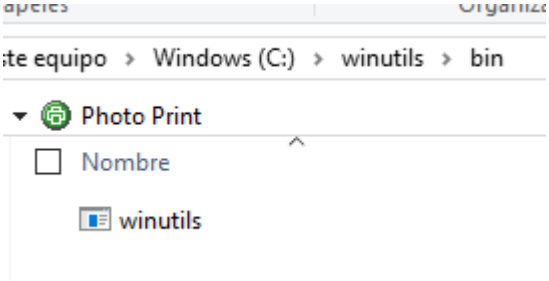
Para utilizar la librería de “mllib” de Spark toca primero importar estas dependencias:

```
<dependencies>
    <dependency>
        <groupId>nz.ac.waikato.cms.weka</groupId>
        <artifactId>weka-stable</artifactId>
        <version>3.8.0</version>
    </dependency>
    <dependency>
        <groupId>org.scala-lang</groupId>
        <artifactId>scala-library</artifactId>
        <version>2.10.5</version>
        <type>jar</type>
    </dependency>
    <dependency>
        <groupId>org.apache.spark</groupId>
        <artifactId>spark-core_2.10</artifactId>
        <version>1.5.2</version>
        <type>jar</type>
    </dependency>
    <dependency>
        <groupId>org.apache.spark</groupId>
        <artifactId>spark-mllib_2.10</artifactId>
        <version>1.3.0</version>
        <type>jar</type>
    </dependency>
</dependencies>
```

Ya con las dependencias necesarias se procede a importar las clases necesarias como se muestra en la imagen de abajo.

```
import java.util.Arrays;
import java.util.HashMap;
import java.util.Map;

import org.apache.spark.mllib.linalg.SparseVector;
import org.apache.spark.mllib.linalg.Vector;
import org.apache.spark.mllib.tree.impl.DecisionTreeMetadata;
import scala.Tuple2;

import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaPairRDD;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.mllib.regression.LabeledPoint;
import org.apache.spark.mllib.tree.DecisionTree;
import org.apache.spark.mllib.tree.model.DecisionTreeModel;
import org.apache.spark.mllib.util.MLUtils;
```

Para que esta herramienta funcione correctamente toca descargar el programa “winutils” que contiene un complemento de Hadoop (para sistemas operativos de Windows).



Luego de descargarla se configura en las variables de entorno del sistema operativo para que se pueda reconocer.

Todo lo anterior mencionado se realiza ya que el modelo de clasificación no puede iniciar y no se coloca esta propiedad en el código.

```
public static void main(String[] args) {

    System.setProperty("hadoop.home.dir", "C:\\winutils\\");
```

Procedemos a crear el SparkConfiguration colocando un nombre y un nodo maestro, en este caso se coloca local y a partir de esto creamos el JavaSparkConfiguration.

```
// Crear la configuración de spark
SparkConf sparkConf = new SparkConf().setAppName("SparkModeloClasificacion").setMaster("local");
JavaSparkContext jsc = new JavaSparkContext(sparkConf);
```

Para poder entrenar este modelo de clasificación con un conjunto de datos, se utiliza un tipo de formato de texto llamado “libsvm”, el cual tiene la siguiente estructura:

```
0 1:18.26 2:57.59 3:221.26
1 1:16.2 2:56.03 3:371.41
1 1:21.84 2:49.98 3:562.88
1 1:20.27 2:46.82 3:441.71
1 1:20.66 2:50.86 3:495.28
1 1:18.82 2:43.28 3:615.91
1 1:16.74 2:46.95 3:484.65
0 1:21.05 2:59.68 3:453.45
1 1:15.18 2:51.94 3:220.73
0 1:21.94 2:58.15 3:574.35
1 1:20.83 2:56.77 3:521.89
1 1:15.48 2:51.79 3:353.86
```

Como se puede observar la primera columna se diferencia del resto ya que esta es la respuesta
que depende de las otras columnas, solo puede ser respuesta binaria ya en este caso cómo se trabajó con una enfermedad inventada (la enfermedad X) entonces si se cumple con una condición de temperatura, humedad e intensidad de la luz que vendrían a ser las otras columnas, entonces la respuesta sería 1 si tiene la enfermedad o en caso contrario 0 si no la tiene (las respuestas no solo pueden ser binarias, esto se puede cambiar modificando una característica del código que se mostrará más adelante). Este tipo de archivos solo maneja números enteros o flotantes.
Una vez ya hecho el conjunto de entrenamiento, se coloca la dirección en donde se encuentra, para que se pueda pasar a un formato llamado JavaRDD<LabeledPoint> para que el sistema lo pueda manipular.

```
//Cargar el dataset de un archivo libsvm
String datapath = "C:\\Users\\Dell\\Desktop\\NovenoSemestre\\PGR1\\weka\\target\\enfermedadX.txt";
//Parsear el archivo a RDD
JavaRDD<LabeledPoint> data = MLUtils.loadLibSVMFile(jsc.sc(), datapath).toJavaRDD();
```

Para entrenar el modelo, en este caso se utilizó el 70% de los datos del conjunto anteriormente mencionado y el 30% restante se utiliza para la evaluación de este, estos parámetros se pueden cambiar a gusto del desarrollador modificando el porcentaje en el arreglo llamado “splits”.

```
//Dividir el dataset 70% para entrenamiento y 30% para pruebas
JavaRDD<LabeledPoint>[] splits = data.randomSplit(new double[]{0.7, 0.3});
JavaRDD<LabeledPoint> trainingData = splits[0];
JavaRDD<LabeledPoint> testData = splits[1];
```

El siguiente paso es definir las propiedades del modelo árbol de decisión, la primera característica se llama “numClasses” la cual se encarga del total de respuestas que va a tener el conjunto de datos de entrenamiento, como se dijo anteriormente, esta variable es igual a 2 ya que queremos manejar un respuesta binaria. La siguiente característica

es “categoricalFeaturesInfo” la cual se encarga de definir cuáles datos va a ser categóricos o continuos, como en el caso de estudio todas las variables (temperatura, humedad e intensidad de la luz) son continuas, entonces el mapa se deja vacío, pero si se quiere establecer una variable categórica entonces se le agrega al mapa como llave la posición que le corresponde a la variable y como valor el número de posibles valores que puede ser, por ejemplo, si se quiere colocar la temperatura como categórica, entonces se coloca la posición 1 y que solo puede ser 3 valores: categoricalFeaturesInfo.put(1,3);
La siguiente característica llamada “maxDepth” se encarga de la profundidad del árbol de decisión (cuantas generaciones), si el árbol es más profundo, será más robusto al momento de hacer predicciones, pero en consecuencia será más complicado de entrenar y consumirá más recursos.
La siguiente característica se llama “impurity”, el cual hace referencia a la impuridad (modelo estadístico) con la cual se va a tratar el modelo, esto sirve para reducir el error de las predicciones y que las respuestas del modelo no sea obra del azar.
La última característica llamada “maxBins” se encarga de discretizar las variables continuas,
esto quiere decir que si la mayoría de las variables son continuas al modelo le queda muy complicado entrenarse, por lo que el número que posee esta variable representa el total de percentiles por el cual se dividió el conjunto de entrenamiento.

```
// Número de respuestas a predecir en este caso (0 ó 1)
int numClasses = 2;
//Si el mapa queda vacio se da por entender que todas las caracteristicas o atributos son continuos
Map<Integer, Integer> categoricalFeaturesInfo = new HashMap<>();
//Impuridad de nodos de clasificación
String impurity = "gini";
//Profundidad del árbol de decisión
int maxDepth = 5;
//Número de contenedores utilizados al discretizar las funciones continuas.
int maxBins = 1000;
```

Ya con estas características configuradas se procede a entrenar el modelo.

```
//Entrenar el modelo de clasificación
DecisionTreeModel model = DecisionTree.trainClassifier(trainingData, numClasses, categoricalFeaturesInfo, impurity, maxDepth, maxBins);
```

Cuando el modelo terminó de entrenarse se procede a evaluarlo, lo que se hace en este caso es utilizar el 30% de los datos restantes, pasandole las características, sin las respuestas como parámetro para que el modelo haga las predicciones y acto seguido compara el total de aciertos que obtuvo con el conjunto de entrenamiento, para tener un porcentaje de acierto.

```
// Evalua el modelo entrenado, comparando la predicción con la respuesta real
JavaPairRDD<Object, Object> predictionAndLabel = testData.mapToPair(p -> new Tuple2<>(model.predict(p.features()), p.label()));
//Calculo de porcentaje de error
double testErr = predictionAndLabel.filter(pl -> !pl._1().equals(pl._2())).count() / (double) testData.count();
System.out.println("Test Error: " + testErr);
System.out.println("Learned classification tree model:\n" + model.toDebugString());
```

Como se puede observar el porcentaje de error es bastante bajo.

```
Test Error: 0.060518115312086236
```

Ya con el modelo entrenado y evaluado, procedemos a guardarlo otorgándole una dirección.

```
//Guardar el modelo
model.save(jsc.sc(), path: "C:\\Users\\Dell\\Desktop\\NovenoSemestre\\PGR1\\weka\\target\\SparkModeloClasificacion");
```

Quedará guardado como una carpeta.

Para cargar el modelo.

```
//Cargar el modelo
DecisionTreeModel sameModel = DecisionTreeModel.load(jsc.sc(), s: "C:\\Users\\Dell\\Desktop\\NovenoSemestre\\PGR1\\weka\\target\\SparkModeloClasificacion");
```

Para finalizar, se procederá a probar que el modelo se cargó correctamente, con un vector el cual poseerá los 3 valores anteriormente mencionados (temperatura, humedad e intensidad de la luz) para que el modelo prediga si con esos datos hay riesgo de enfermedad.

```
double[] vector = {22.0,40.0,390.0};
int[] index = {0,1,2};
Vector v = new SparseVector( size: 3,index,vector);
System.out.println("El modelo predijo que: "+sameModel.predict(v));
```

Y como se puede observar, el modelo funciona correctamente, ya que predijo que con los datos entregados no hay riesgo de enfermedad.

```
El modelo predijo que: 0.0
```

5.4.5. Guía para la creación de nuevas aplicaciones de Spark Streaming que lean datos de telemetría en tiempo real.

Cada vez que se desee leer en tiempo real un dato de telemetría nuevo, por ejemplo “Intensidad de la luz”, es necesario crear una aplicación de Apache Spark Streaming para poder definir así el tratamiento a este nuevo tipo de datos.

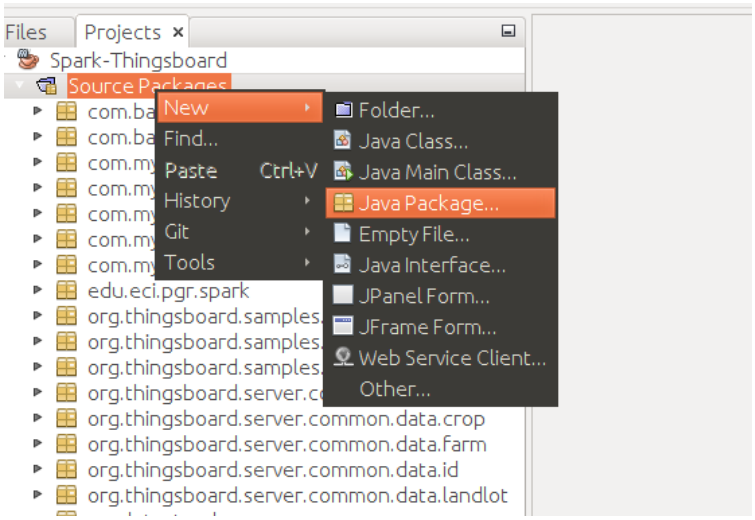
Apache Spark cuenta con una herramienta para leer datos desde tópicos de kafka en tiempo real llamada Spark Streaming. En esta guía se implementará una aplicación que lea datos de la intensidad de la luz de un tópico de Kafka en tiempo real utilizando Spark Streaming, y posteriormente guarde los datos en redis para que la aplicación de temperatura y las reglas puedan tener acceso a los mismos.

Como pre-requisito se debe tener en kafka un tópico con el nombre “light” (Ver Capítulo 5.2 - Manual de instalación, Sección Instalación de Kafka, Punto 9).

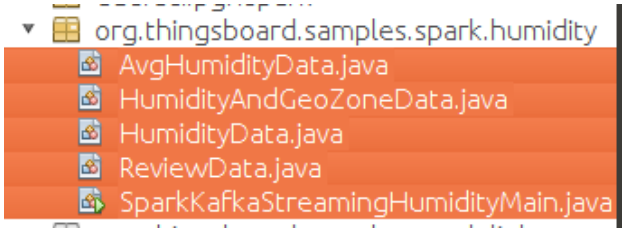
1. En caso de no tener el código fuente del proyecto de las aplicaciones de Apache Spark para Thingsboard, puede clonar el proyecto con el comando:

git clone *https://github.com/LIS-ECI/thingsboard-spark-backend*

2. Abrir el proyecto en un editor, en este caso se utilizará Netbeans, posteriormente se crea un paquete con el nombre org.thingsboard.samples.spark.light



3. Para construir la nueva aplicación nos basaremos en la aplicación de temperatura ya creada previamente, y en las 5 clases que la componen (AvgHumidityData, HumidityAndGeoZoneData, HumidityData, ReviewData y SparkKafkaStreamingHumidityMain)



Pero en este caso se deben crear las 5 nuevas clases en el paquete previamente creado y cambiar sus nombres para relacionarlos con la Intensidad de la luz en este caso “light”, en esta guía se proponen los siguientes nombres:

- AvgLightData.java
- LightAndGeoZoneData.java
- LightData.java
- ReviewData.java
- SparkKafkaStreamingLightMain.java

Respectivamente

4. Se modifican las clases HumidityData, HumidityAndGeoZoneData y AvgHumidityData de forma que:

- los nombres de la clase y de sus respectivos archivos (.java) sean cambiados por LightData, LightAndGeoZoneData y AvgLightData
- Las clases deben quedar tal y como se muestra en las siguientes imágenes, haciendo un importante énfasis en las clases LightData, LightAndGeoZoneData, ya que en estas se debe cambiar las menciones de humedad por “light”.

AvgLightData.java

```java
@Data
@NoArgsConstructor
@AllArgsConstructor
public class AvgLightData implements Serializable {

    private double value;
    private int count;

    public AvgLightData(double value) {
        this.value = value;
        this.count = 1;
    }

    public double getAvgValue() {
        return value / count;
    }

    public int getCount(){
        return count;
    }

    public static AvgLightData sum(AvgLightData a, AvgLightData b) {
        return new AvgLightData(a.value + b.value, a.count + b.count);
    }

}
```

LightData.java

```java
package org.thingsboard.samples.spark.light;

import org.thingsboard.samples.spark.humidity.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

/**
 * Created by Valerii Sosliuk on 10/28/2017.
 */
@Data
@NoArgsConstructor
@AllArgsConstructor
public class LightData {

    private double light;

}
```

LightAndGeoZoneData.java

```
13  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express o
14  * See the License for the specific language governing permission
15  * limitations under the License.
16  */
17 package org.thingsboard.samples.spark.light;
18
19 import org.thingsboard.samples.spark.humidity.*;
20 import lombok.AllArgsConstructor;
21 import lombok.Data;
22 import lombok.NoArgsConstructor;
23
24 import java.io.Serializable;
25
26 /**
27  */
28 @Data
29 @NoArgsConstructor
30 @AllArgsConstructor
31 public class LightAndGeoZoneData  implements Serializable {
32
33     private String deviceId;
34     private double light;
35     private int count;
36
37 }
```

7. Para la clase *SparkKafkaStreamingLightMain* lo recomendado es basarse en la clase *SparkKafkaStreamingHumidityMain* y utilizar las clases propias del paquete, otros cambios clave son:

- Línea 49 *private static final String Topic="humidity";* cambiar el tópico por *“light”*
- Línea 52 *public static final String APP_NAME = "Spark Humidity";* cambiar el nombre de la aplicación por “*Spark Light";*
- Línea 98 hasta línea 109, Cambiar los tipos de de datos mofidificados anteriormente de forma que no se usen los de humedad si no los de “light”de forma que quede como en la siguiente imagen

```
// Map incoming JSON to LightAndGeoZoneData objects
JavaRDD<LightAndGeoZoneData> lightRdd = rdd.map(new LightStationDataMapper());
// Map WindSpeedAndGeoZoneData objects by GeoZone

JavaPairRDD<String, AvgLightData> lightByZoneRdd = lightRdd.mapToPair(d -> new Tuple2<>(d.getDeviceId(),
        new AvgLightData(d.getLight())));
// Reduce all data volume by GeoZone key
lightByZoneRdd = lightByZoneRdd.reduceByKey((a, b) -> AvgLightData.sum(a, b));
// Map <GeoZone, AvgWindSpeedData> back to WindSpeedAndGeoZoneData
List<LightAndGeoZoneData> aggData = lightByZoneRdd.map(t ->
        new LightAndGeoZoneData(t._1, t._2.getAvgValue(),t._2.getCount())).collect();
// Push aggregated data to ThingsBoard Asset
//restClient.sendTelemetryToAsset(aggData);
if (!aggData.isEmpty()) {
    JavaRDD<LightAndGeoZoneData> telemetryData = ssc.sparkContext().parallelize(aggData);
    reviewData.analizeTelemetry(telemetryData,Topic,rulesEngine);
}
```

- Modificar la clase HumidityStationDataMapper cambiandole el nombre por LightStationDataMapper, tambien haciendo que implemente Function<ConsumerRecord<String, String>, LightAndGeoZoneData>
- Dentro de la clase modificar el método Call de forma que se vea como en la imagen

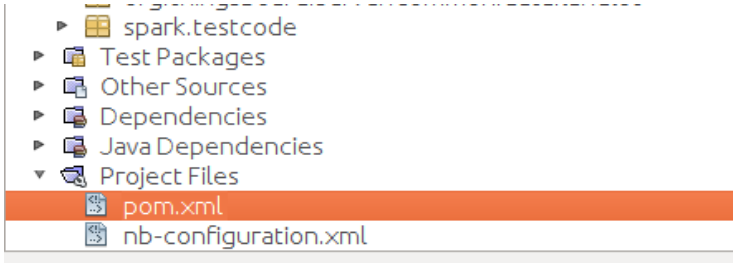
```
@Override
public LightAndGeoZoneData call(ConsumerRecord<String, String> record) throws Exception {
    return mapper.readValue(record.value(), LightAndGeoZoneData.class);
}
```

8. Modificar la clase ReviewData del paquete creado haciendo que el método analizeTelemetry, reciba por parámetro un JavaRDDLightAndGeoZoneData> en vez de un (JavaRDD<HumidityAndGeoZoneData>

- Línea 58 hmap = telemetryData.mapToPair((HumidityAndGeoZoneData telemetryData1) -> … , utilizar el tipo LightAndGeoZoneData
- Línea 60 return new Tuple2(idLandlot, telemetryData1.getHumidity()); cambiar el get por getLight

Una vez realizados estos pasos se procede a guardar el programa y verificar que no hayan errores en el código

9. Ir a Project Files, abrir el contenedor y abrir el archivo pom.xml



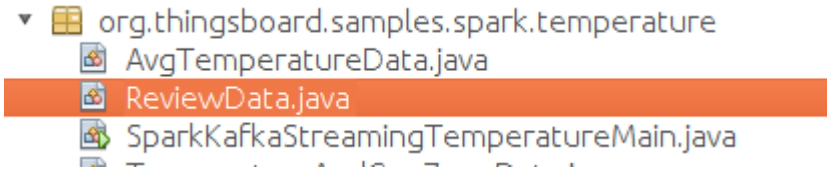
Al final del xml se puede ver un tag <execution> … </execution>, se debe crear otro, en el cual se cambie el <id>make-assembly-light</id>, tambien el <mainClass> por <mainClass>org.thingsboard.samples.spark.temperature.SparkKafkaStreamingLightMain< /mainClass> y el <finalName> por <finalName>Spark-Light-KafkaStreaming</finalName>

Lo cual tendrá como resultado final

```
<execution>
    <id>make-assembly-light</id>
    <phase>package</phase>
    <goals>
        <goal>single</goal>
    </goals>
    <configuration>
        <archive>
            <manifest>
                <mainClass>org.thingsboard.samples.spark.temperature.SparkKafkaStreamingLightMain</mainClass>
            </manifest>
        </archive>
        <descriptorRefs>
            <descriptorRef>jar-with-dependencies</descriptorRef>
        </descriptorRefs>
        <finalName>Spark-Light-KafkaStreaming</finalName>
        <appendAssemblyId>false</appendAssemblyId>
    </configuration>
</execution>
```

Modificar la aplicación de temperatura para que lea los datos de Intensidad de la luz, “light”, y posteriormente los pase a las reglas

10. Se procede a abrir la clase *ReviewData* del paquete org.thingsboard.samples.spark.temperature



11. Antes de la creación del HashMap dataApplications, se debe obtener el valor desde redis tal y como en la siguiente línea de código (puede utilizar la obtención de datos de humedad como ejemplo y editar las variables).

```
String templight = ExternalMethods.getValueOfRedis("light", idLandlot);
Double lightyData = 0.0;
if (templight != null) {
    lightyData = Double.parseDouble(templight);
}
```

12. Añadir el dato al HashMap para que sea accesible por las reglas

```
dataApplications.put("temperatureData", String.valueOf(temperatureData));
dataApplications.put("lightData", String.valueOf(lightyData));
dataApplications.put("idLandlot", idLandlot);
```

13. Compilar el proyecto, con el comando

mvn clean package, en la raíz del proyecto

```
/thingsboard-spark-backend$ mvn clean package
```

14. Una vez haya terminado de compilar se debe acceder a la carpeta target

```
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 05:04 min
[INFO] Finished at: 2018-07-14T17:39:56-05:00
[INFO] Final Memory: 85M/672M
[INFO] ------------------------------------------------------------------------
```

cd target/

y Verificar que la aplicación (.jar) haya sido creada

ls

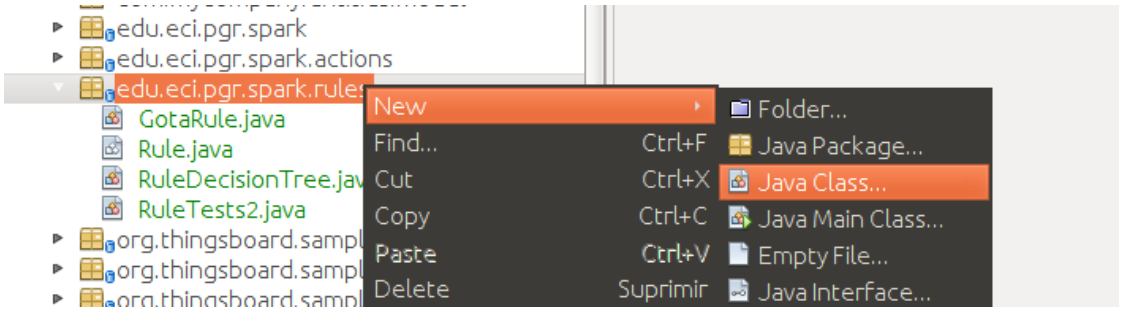
```
archive-tmp          maven-status                          Spark-Thingsboard-1.0-SNAPSHOT-fat.jar
classes              Spark-Humidity-KafkaStreaming.jar     Spark-Thingsboard-1.0-SNAPSHOT.jar
generated-sources    Spark-Light-KafkaStreaming.jar
maven-archiver       Spark-Temperature-KafkaStreaming.jar
```

Ahora las aplicaciones se encuentran listas para ser ejecutadas en el clúster.

5.4.5. Guía para la creación de nuevas reglas en Apache Spark.

5.4.5.1. Creación de una regla:

Para crear una regla se debe ir al paquete edu.edi.pgr.spark.rules y crear una nueva clase que será la Regla



El nombre de la regla es a libre elección, en este ejemplo se llamará *RuleTestTutorial*

- Esta clase debe extender de la clase Rule e implementar Serializable
- Se debe crear un atributo de tipo lista de String, llamado types_Crops, el cual hace referencia a los cultivos a los cuales la regla puede ser aplicada El atributo anterior debe tener su respectivo getter y setter
- Se debe crear un constructor que no recibe parámetros, en el cual se inicializa la lista y se le agregan los Strings que denotan los nombres de los cultivos que aplica
- Debe crear un método public void execute(HashMap<String,String> data, DecisionTreeModel model){ } el cual va a ser el primer método en ser llamado y dónde estará toda la lógica de la regla.

En este caso el esqueleto de la nueva regla es:

RuleTestTutorial.java

Source History

```
/**
 *
 * @author cristian
 */
public class RuleTestTutorial extends Rule implements Serializable{

    List<String> types_Crops;

    public List<String> getTypes_Crops() {
        return types_Crops;
    }

    public void setTypes_Crops(List<String> types_Crops) {
        this.types_Crops = types_Crops;
    }

    public RuleTestTutorial(){
        types_Crops= new ArrayList<>();
        types_Crops.add("Potato");
    }

    @Override
    public void execute(HashMap<String,String> data,DecisionTreeModel model){
        System.out.println("RuleTestTutorial Is Working!!");
    }
}
```

5.4.5.2. Integrar a la regla funcionalidades de Spark.

Existe una clase en el paquete org.thingsboard.samples.spark.util llamada ExternalMethods la cual contiene diferentes implementaciones de métodos estáticos para acceder a fuentes de datos externas a las entrantes a la aplicación de spark.

5.4.5.2.1 Acceder a los datos de Cassandra:

Existe un paquete llamado edu.eci.pgr.cassandra.java.client.repository, en donde actualmente existen 3 clases, LandlotRepository, KeyspaceRepository, FarmRepository, cada uno diseñado específicamente para contener métodos que permitan hacer consultas sobre las tablas de Cassandra.
Actualmente sólo poseen métodos para obtener objetos por el Id, pero está abierto a que se puedan realizar consultas más complejas en el futuro.

La clase ExternalMethods utiliza estos repositorios para realizar consultas a cassandra e implementar nuevos métodos a partir de los existentes, por ejemplo contiene un método llamado getCropNameCassandra(String idLandlot) que retorna un String que contiene el nombre del cultivo a partir del Id del lote

En este caso la regla anteriormente creada accederá al nombre del cultivo a través de cassandra

```
@Override
public void execute(HashMap<String,String> data,DecisionTreeModel model){
    System.out.println("RuleTestTutorial Is Working!!");
    String idLandlot = data.get("idLandlot");
    String cropName = ExternalMethods.getCropNameCassandra(idLandlot);
    System.out.println("This is the crop name!: "+cropName);
}
```

Ya que el HashMap trae el Id del lote, (y también el nombre del cultivo pero para efectos del ejercicio supondremos que no), se accederá a Cassandra para obtener el nombre del cultivo, en este caso a la clase ExternalMethods y al método getCropNameCassandra(String idLandlot)

5.4.5.2.2 Acceder a los datos de Redis:

Conforme a la forma de utilizar redis por las aplicaciones se tienen 2 métodos principales, uno para **guardar datos** saveToRedis(String key, String idLandlot, String data) en donde se mezcla la llave con el id del lote para generar una nueva llave única, y en donde se pasa por parámetro el valor a gurdar en dicha llave

Para **acceder a los datos** existe en método getValueOfRedis(String key, String idLandlot), en donde recibe por parámetro la llave y el id del lote para retornar el valor almacenado en dicha llave

En este caso la regla anteriormente creada consultará el último dato de humedad del lote.

```java
@Override
public void execute(HashMap<String,String> data,DecisionTreeModel model){
    System.out.println("RuleTestTutorial Is Working!!");

    String idLandlot = data.get("idLandlot");
    String lastHumidity = ExternalMethods.getValueOfRedis("humidity", idLandlot);
    System.out.println("This is the last humidity data received!: "+lastHumidity);
```

5.4.5.2.3 Acceder a los datos de MongoDB:

MongoDB es principalmente usado para almacenar Fincas, Lotes y Dispositivos con propiedades de georeferenciación, y además para obtener los Spark Devices los cuales son los principales agentes son los para el envío de alertas a la plataforma Thingsboard

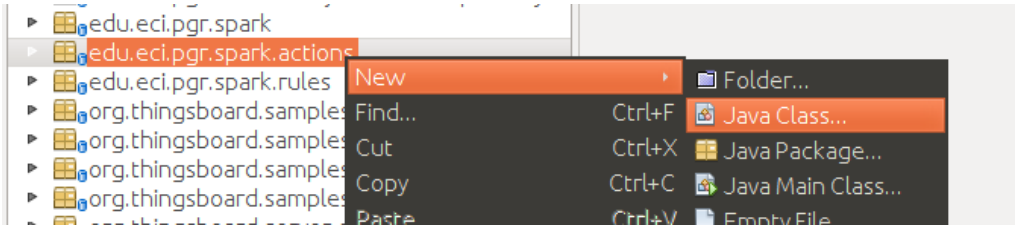
La clase principal es MongoDBSpatial a través de ella se puede acceder a las distintas colecciones de Mongo y obtener información de diferentes entidades anteriormente mencionadas.

Para acceder a las entidades anteriormente mencionadas se tienen implementaciones por cada una, MongoDBSpatialLandlot para los lotes, MongoDBSpatialSpark para los dispositivos de tipo Spark, MongoDBSpatialFarm para las fincas, MongoDBSpatialDevice para los dispositivos, además en la clase MongoDBSpatial también se cuenta con métodos para consultar información que reside en MongoDB.

En este caso la regla ejecutará una acción que accede los datos de la ubicación del lote y los imprimirá por pantalla.

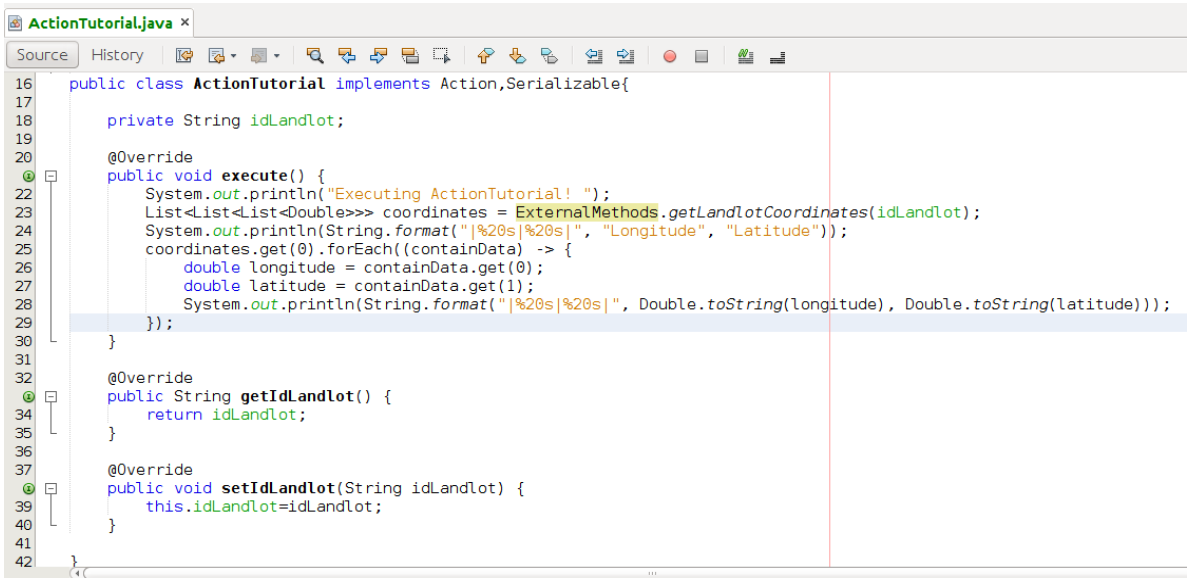
5.4.5.2.4. Creación de acciones:

Para crear una acción se debe crear una clase en el paquete edu.eci.pgr.spark.actions, en este caso la acción tendrá como nombre ActionTutorial, esta acción debe implementar Action y Serializable.



En el método execute() se utilizará el método getLandlotCoordinates(String idLandlot) de la clase ExternalMethods para traer las coordenadas del lote desde MongoDB.

En la imagen se muestra el esqueleto de la acción creada



Para que la regla tenga acciones se debe crear un atributo de tipo List<Action>, y en el constructor debe instanciarse y agregarse las acciones a ejecutar, cabe destacar que es el esquema propuesto, sin embargo queda abierto para que se puedan instanciar las acciones como se desee.

```
public RuleTestTutorial(){
    types_Crops= new ArrayList<>();
    types_Crops.add("Potato");
    actions = new ArrayList<>();
    actions.add(new ActionTutorial());

}
```

En este ejemplo la regla tendrá en cuenta que si el dato de intensidad de la luz es mayor a 50, ejecutará las acciones asociadas a la regla.

```
@Override
public void execute(HashMap<String,String> data,DecisionTreeModel model){
    System.out.println("RuleTestTutorial Is Working!!");
    String idLandlot = data.get("idLandlot");
    String lastHumidity = ExternalMethods.getValueOfRedis("humidity", idLandlot);
    System.out.println("This is the last humidity data received!: "+lastHumidity);
    String cropName = ExternalMethods.getCropNameCassandra(idLandlot);
    System.out.println("This is the crop name!: "+cropName);
    double light = data.get("lightData");
    if (light>50){
        actions.forEach((a) -> {
            a.setIdLandlot(idLandlot);
            a.execute();
        });
    }

}
```

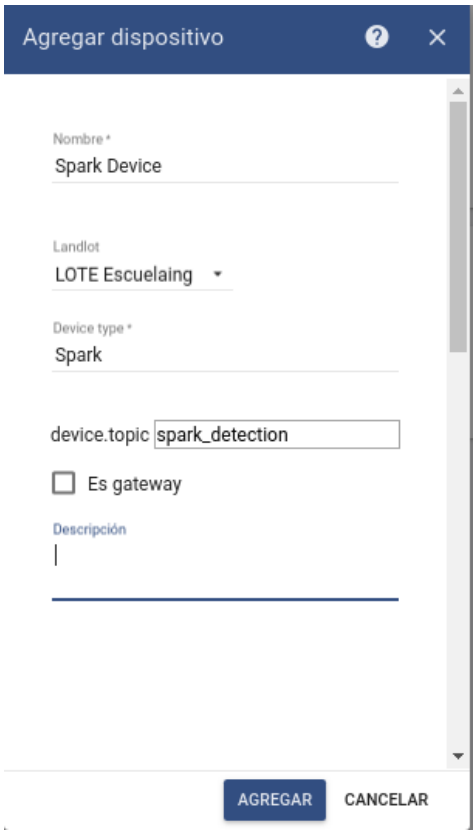
5.4.5.2.5 Generar y notificar Alertas a Thingsboard:

Para que una alarma sea recibida por Thingsboard hay que tener creado un dispositivo de tipo Spark sobre un cultivo, y configurada una regla dentro del sistema.

El dispositivo de tipo Spark se encargará de recibir datos que pueden activar una alarma sobre un cultivo.

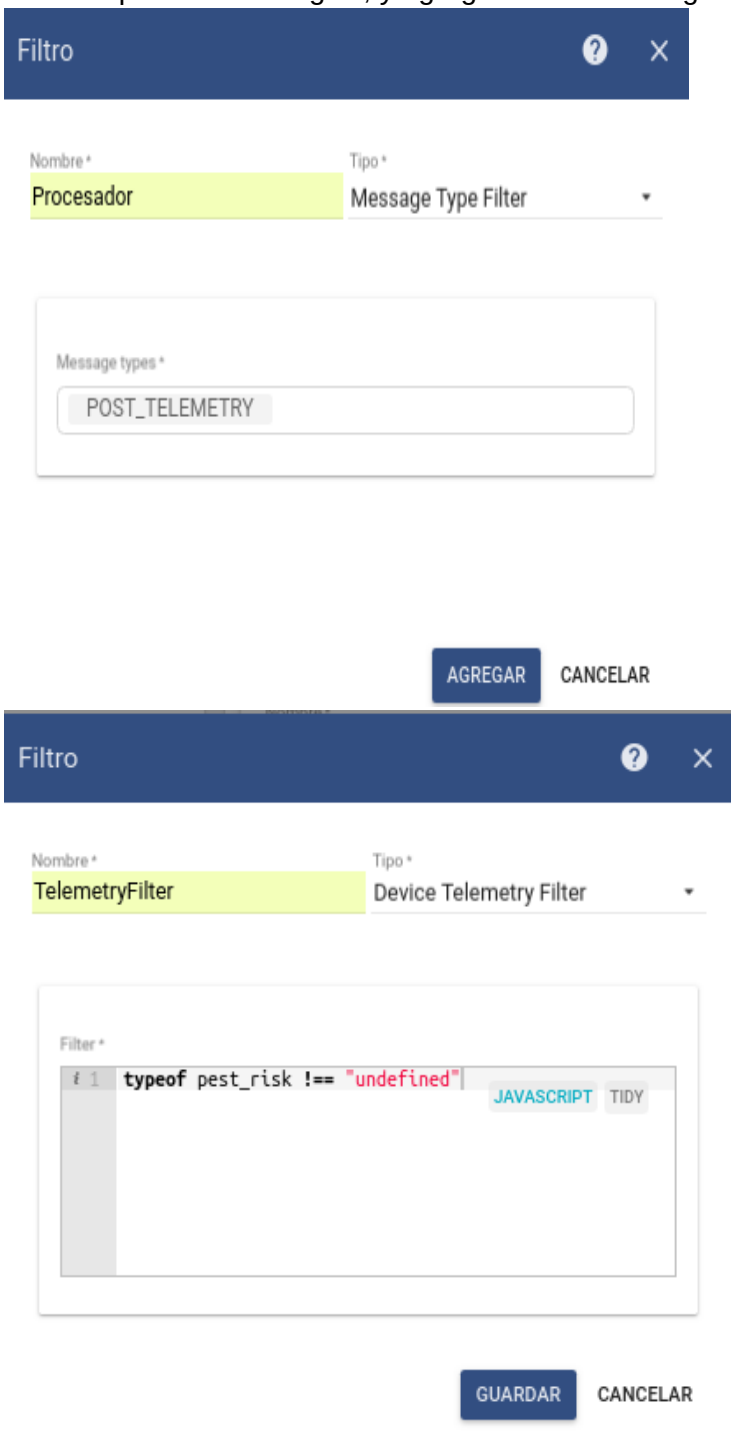
1. Ingresar a la plataforma de Thingsboard

2. Ir a la pestaña de dispositivos y crear un dispositivo, este dispositivo debe tener como Device_type: Spark, y se le debe asignar un token que servirá para identificarlo dentro de la aplicación de Spark.

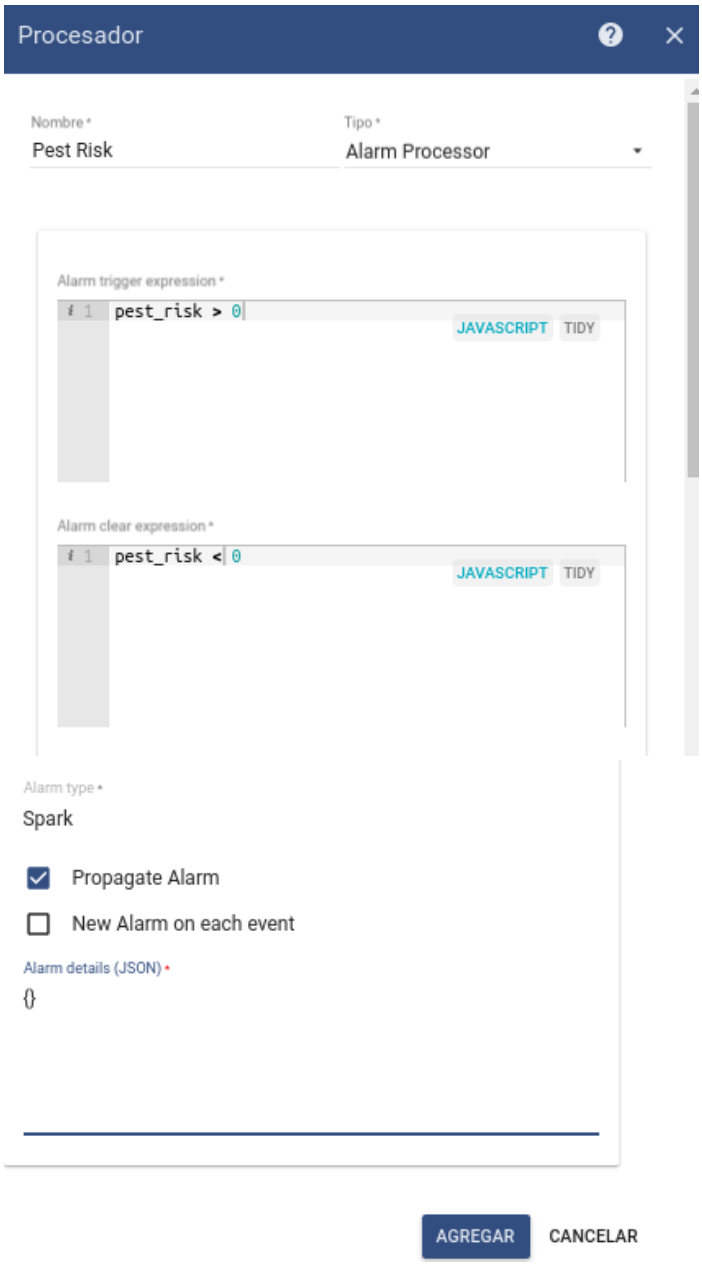


Una vez hecho esto, ese dispositivo será el encargado de recibir los datos desde la aplicación de Spark, ahora se creará una regla que dispare la alarma que lanzaremos desde Spark.

3. Ir a la pestaña de Reglas, y agregar una nueva regla con dos filtros



y cuyo procesador sea



En este caso cuando sea detectado en Spark un riesgo de plaga se enviará un dato > 0 para que la regla se active y genere una alarma.

4. Una vez creada la regla se procede a activarla

Ahora en Spark se enviará un dato de telemetría al dispositivo para que se genere la alarma, para esto se utilizarán 2 métodos, el primero es getTokenSpark el cual retorna el token del dispositivo Spark en Thingsboard para así poderle enviar un dato de telemetría por Mqtt

Una vez obtenido el token se puede hacer uso del método sendTelemetryDataToThingsboard(String token, String key, double value); en el cual el "key" es la llave con la cual se va a identificar el dato de telemetría a enviar, en este caso se configuró con "pest_risk" ya que la regla está configurada para analizar los datos de telemetría entrantes con este valor.

```
String token = ExternalMethods.getTokenSpark(getIdLandlot(), "spark_detection");
try {
    ExternalMethods.sendTelemetryDataToThingsboard(token, "pest_risk", 1);
}
catch (Exception ex) {
    Logger.getLogger(ActionSendAlert.class.getName()).log(Level.SEVERE, null, ex);
}
```

6. Prueba de concepto

Se analizaron 2 enfermedades:

- El tizón tardío producido por el hongo “Phytophthora infestans”, el cual afecta más comúnmente a los cultivos de papa y tomate dañando las hojas, el tallo y el fruto. Las condiciones favorables para que se manifieste este hongo son:
  - temperatura>=10° con una humedad relativa>= 90%, durante 11 horas, durante 2 dias seguidos [2]
- Enfermedad X, es una enfermedad inventada que sirve para ejemplificar el uso de algoritmos de clasificación. Para que se den las condiciones favorables se debe cumplir esta inecuación

$$\frac{Intensidad\ de\ la\ luz}{Humedad} - 12 - temperatura > 0$$

Se pueden integrar al sistema distintos tipos de enfermedades o males que pueden ocurrir sobre un cultivo.

Se utilizaron 10 dispositivos cada uno cuenta con 3 sensores, temperatura ambiente, Intensidad de la luz y humedad, los cuales envían datos cada 10 minutos.

En la plataforma Thingsboard se creó una finca en la UDCA, con 3 lotes cada lote tiene un cultivo de Papa, y cada cultivo dispone de 3 dispositivos físicos los cuales se comunican a través del protocolo MQTT a una raspberry para que posteriormente se envien los datos a la plataforma Thingsboard.

También se dispone de un dron el cual se encarga de tomar fotos multiespectrales a los cultivos (RGB, REG, RED, GRE, NIR) y se cargaron a la plataforma de manera que quedan asociadas a los respectivos lotes por medio de un cálculo georeferenciado obtenidos de sus metadatos (longitud, latitud)

Una vez creada la finca, los lotes y sus respectivos dispositivos se crea automáticamente un dashboard con el nombre de la finca, el cual contiene un mapa que permite visualizar en tiempo real el estado de la finca con sus cultivos, detallando en los dispositivos los últimos valores de telemetría que le llegaron.

Se crearon 3 reglas en Thingsboard que toman los datos enviados por los dispositivos y los publican en Kafka, para que sean consultados por las aplicaciones de Apache Spark, las aplicaciones de spark estará cada n segundos revisando sus respectivos tópicos de kafka (el n varía según el tiempo de transmisión que demoran los sensores en reportar los datos de telemetría), una vez estén toda la información almacenada, se procesaran los datos para separarlos por cultivo, se promedian estos datos y se evalúan las reglas definidas por el usuario, en caso de que alguna regla de como resultado una probabilidad de enfermedad, se disparan las acciones asociadas a la regla, las reglas pueden ser algoritmos o modelos de machine learning y pueden utilizar info en cassandra (datos históricos) o mongo (datos georreferenciados) según sea necesario.

En este caso, la regla que hace uso de machine learning (modelo de clasificación) está entrenada con datos sobre la enfermedad X, y en caso de que el cultivo tengo riesgo de la enfermedad, se ejecutan acción que envía por MQTT un dato a Thingsboard que activa una

alarma, la cual se ve posteriormente reflejada en el dashboard (cambiando el color del lote). Para ver con detalle la alarma existe un dashboard alterno el cual detalla el lote afectado y la enfermedad.

En el caso del tizón tardío, la regla que hace la evaluación en caso de que se den las condiciones para que se dé el tizón tardío ejecuta una acción que envía los datos de las coordenadas de la finca afectada a un dron para que posteriormente aplique un fungicida.

La plataforma posteriormente notificará al correo del usuario que el cultivo tiene un riesgo.

Otra funcionalidad que ofrece la plataforma es facilitar la creación de etiquetas que alimentan a un conjunto de entrenamiento para el modelo de clasificación, en esta funcionalidad permite seleccionar entre un rango de fechas para observar el histórico de datos representado por gráficas (uno por cada tipo de dato de telemetría), esto también aplica para visualizar fotos de un cultivo de una fecha en específico sobre un mapa. Con la información anterior el experto puede decidir que tipo de enfermedad puede estar afectando el cultivo seleccionando también una zona que él crea que está afectada sobre las fotos, y escribiendo la etiqueta respectiva.

Pasos para la ejecución de la prueba de concepto:

Cada una de las instrucciones siguientes debe hacerse en un terminal independiente

En caso de utilizar algún modelo de machine learning debe tenerlo almacenado en todos los nodos en donde se vaya a ejecutar algún trabajador de spark y en la misma dirección

En este caso la aplicación de spark está configurada para que el modelo esté alojado en:

"/home/pgr/decision_tree/SparkModeloClasificacion"

agricultura2:

- Correr redis
  cd redis-stable/
  sudo redis-server redis.conf
- Correr Spark Master
  /home/pgr/spark-2.3.1-bin-hadoop2.7/bin/spark-class
  org.apache.spark.deploy.master.Master
- Correr Spark Worker
  /home/pgr/spark-2.3.1-bin-hadoop2.7/bin/spark-class
  org.apache.spark.deploy.worker.Worker spark://10.8.0.17:7077

agricultura1:

- Correr Thingsboard
- Correr Kafka
  sudo /opt/kafka/bin/kafka-server-start.sh /opt/kafka/config/server.properties

- Correr Spark Worker
  /home/pgr/pgr/spark-2.3.1-bin-hadoop2.7/bin/spark-class org.apache.spark.deploy.worker.Worker spark://10.8.0.17:7077

agricultura3:

- Correr Spark Worker
  /home/pgr/spark-2.3.1-bin-hadoop2.7/bin/spark-class org.apache.spark.deploy.worker.Worker spark://10.8.0.17:7077

Envío de las aplicaciones de Spark al clúster

agricultura2:

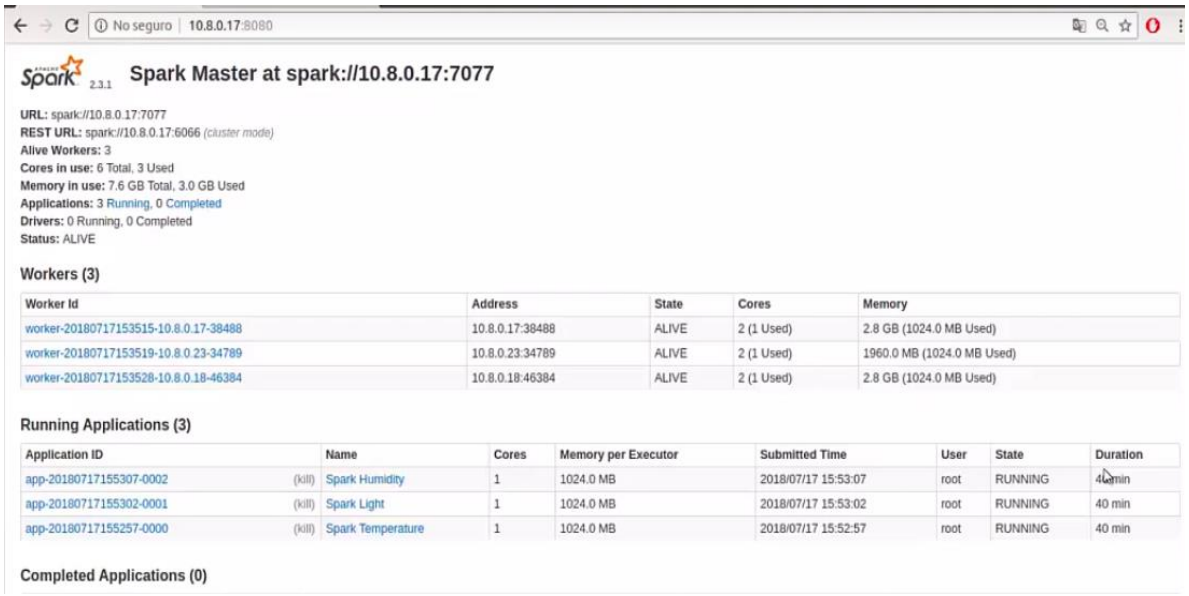
En caso de no tener descargado el código fuente de thingsboard-spark-backend puede descargarlo desde https://github.com/LIS-ECI/thingsboard-spark-backend con el comando

git clone https://github.com/LIS-ECI/thingsboard-spark-backend

- Compilar las aplicaciones de spark :
  cd thingsboard-spark-backend
  mvn package
- Enviar al cluster la aplicación de humedad
  sudo /home/pgr/spark-2.3.1-bin-hadoop2.7/bin/spark-submit --class org.thingsboard.samples.spark.humidity.SparkKafkaStreamingHumidityMain --master spark://10.8.0.17:7077 --conf spark.cores.max=1 /home/pgr/thingsboard-spark-backend/target/Spark-Humidity-KafkaStreaming.jar
- Enviar al cluster la aplicación de intensidad de la luz
  sudo /home/pgr/spark-2.3.1-bin-hadoop2.7/bin/spark-submit --class org.thingsboard.samples.spark.light.SparkKafkaStreamingLightMain --master spark://10.8.0.17:7077 --conf spark.cores.max=1 /home/pgr/thingsboard-spark-backend/target/Spark-Light-KafkaStreaming.jar
- Enviar al cluster la aplicación de temperatura
  sudo /home/pgr/spark-2.3.1-bin-hadoop2.7/bin/spark-submit --class org.thingsboard.samples.spark.temperature.SparkKafkaStreamingTemperatureMain --master spark://10.8.0.17:7077 --conf spark.cores.max=1 /home/pgr/thingsboard-spark-backend/target/Spark-Temperature-KafkaStreaming.jar

Puede cambiar la cantidad de cores que utilizan las aplicaciones cambiando el parámetro en los comandos “ spark.cores.max=1 “ por otro número

Para acceder a la página principal del nodo master de spark debe ir a 10.8.0.17:8080 y aparecerá una pantalla como la siguiente:


No seguro | 10.8.0.17:8080
Spark 2.3.1
Spark Master at spark://10.8.0.17:7077
URL: spark://10.8.0.17:7077
REST URL: spark://10.8.0.17:6066 (cluster mode)
Alive Workers: 3
Cores in use: 6 Total, 3 Used
Memory in use: 7.6 GB Total, 3.0 GB Used
Applications: 3 Running, 0 Completed
Drivers: 0 Running, 0 Completed
Status: ALIVE
Workers (3)
Worker Id | Address | State | Cores | Memory
worker-20180717153515-10.8.0.17-38488 | 10.8.0.17:38488 | ALIVE | 2 (1 Used) | 2.8 GB (1024.0 MB Used)
worker-20180717153519-10.8.0.23-34789 | 10.8.0.23:34789 | ALIVE | 2 (1 Used) | 1960.0 MB (1024.0 MB Used)
worker-20180717153528-10.8.0.18-46384 | 10.8.0.18:46384 | ALIVE | 2 (1 Used) | 2.8 GB (1024.0 MB Used)
Running Applications (3)
Application ID | Name | Cores | Memory per Executor | Submitted Time | User | State | Duration
app-20180717155307-0002 (kill) | Spark Humidity | 1 | 1024.0 MB | 2018/07/17 15:53:07 | root | RUNNING | 40 min
app-20180717155302-0001 (kill) | Spark Light | 1 | 1024.0 MB | 2018/07/17 15:53:02 | root | RUNNING | 40 min
app-20180717155257-0000 (kill) | Spark Temperature | 1 | 1024.0 MB | 2018/07/17 15:52:57 | root | RUNNING | 40 min
Completed Applications (0)

7. Conclusiones

A partir del proceso de investigación, se logró comprender el caso de estudio y extender las funcionalidades que “Thingsboard” ofrece para adecuarlas de acuerdo a los objetivos del proyecto.

Se logró evidenciar la importancia que tiene el manejo de la información para el sector agrícola en Colombia, con el fin de mejorar el desarrollo de esta práctica.

Con el uso de las herramientas expuestas se pueden llegar a detectar amenazas en tiempo real, o por medio de históricos lograr predecir comportamientos que pueden afectar a un cultivo. No solo permite usar machine learning con datos numéricos, sino que existe la posibilidad de implementar modelos que analizan imágenes ya que estas también poseen la noción del tiempo ya que el modelo de clasificación que se utilizó no cuenta con este.

Haciendo uso de los 5 tipos de imágenes (RED, RGB, REG, GRE, NIR) tomadas por un dron, en un punto del cultivo, es posible lograr la detección de enfermedades, ya que, con estas, se pueden realizar cálculos multiespectrales que permiten ver el ciclo de vida que tiene el cultivo y detallar si su crecimiento no está ocurriendo cómo debería ser.

Actualmente la plataforma notifica al usuario cuando algún cultivo sufre un riesgo, pero a partir de esto se le pueden enviar órdenes a entidades IoT para que solucionen el problema sobre el terreno afectado, haciendo uso del modelo de reglas y acciones de Spark.

8. Referencias

1. V. A. A. Espana, A. R. R. Pinilla, P. Bardos, and R. Naidu. Contaminated land in colombia: A critical review of current status and future approach for the management of contaminated sites. Science of the Total Environment, 618:199–209, 2018.
2. I. Iglesias, O. Escuredo, C. Seijo, and J. Mendez. Phytophthora infestans prediction for a potato crop. American journal of potato research, 87(1):32–40, 2010.
3. M. Mazhar Rathore, A. P. (23 de Diciembre de 2015). Urban planning and building smart cities based on the internet of. *Computer Network*
4. RDD Programming Guide - Spark 2.3.1 Documentation. (2018). Retrieved from https://spark.apache.org/docs/latest/rdd-programming-guide.html#resilient-distributed-datasets-rdds
5. Spatial queries. (2018). Retrieved from https://docs.datastax.com/en/datastax_enterprise/4.8/datastax_enterprise/srch/srchSpatialQueries.html
6. GridFS — MongoDB Manual. (2018). Retrieved from https://docs.mongodb.com/manual/core/gridfs/
7. Documentation | Aerospike. (2018). Retrieved from https://www.aerospike.com/docs/guide/geospatial.html