ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

IMPLEMENTACION DE UN PROTOTIPO DE RASTREO DE BICICLETAS CON TECNOLOGÍA SIGFOX

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES

RICARDO JAVIER URBINA OCAMPO

DIRECTOR: Msc. CARLOS ROBERTO EGAS ACOSTA

Quito, marzo 2022

AVAL

Certifico que el presente trabajo fue desarrollado por Ricardo Javier Urbina Ocampo, bajo nuestra supervisión.

Msc. Carlos Egas

DIRECTOR DEL TRABAJO DE TITULACIÓN

DECLARACIÓN DE AUTORÍA

Yo, Ricardo Javier Urbina Ocampo, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración dejo constancia de que la Escuela Politécnica Nacional podrá hacer uso del presente trabajo según los términos estipulados en la Ley, Reglamentos y Normas vigentes.

RICARDO JAVIER URBINA OCAMPO

DEDICATORIA

A mi querida familia.

AGRADECIMIENTO

Quiero agradecer a mis padres, Ricardo y Aide, quienes me han sido un pilar fundamental en mi vida y me han brindado la posibilidad de cumplir con todos mis propósitos que me he propuesto, por todo el cariño que me han brindado.

A mi hermana Sofía por su apoyo incondicional en todos mis proyectos y por estar siempre en los momentos más importantes de mi vida.

Finalmente agradezco a todos mis amigos y familiares que supieron brindarme su apoyo incondicional cuando más se lo requería.

ÍNDICE DE CONTENIDO

RESUMEN

El presente trabajo de titulación muestra un modelo de rastreo de bicicletas con la red Sigfox por medio del uso de nodos de la marca Pycom. El modelo consta de 3 componentes principales que son el componente Sigfox, el componente AWS y la aplicación de usuario; los 3 componentes tendrán su fase de diseño, fase de implementación y fase de pruebas.

El primer capítulo contiene todos los aspectos teóricos del prototipo de rastreo, en el cual se detallan aspectos técnicos de las tecnologías utilizadas en el modelo.

El segundo capítulo detalla el diseño e implementación de los 3 componentes: componente Sigfox, el componente AWS y la aplicación de usuario.

En el tercer capítulo se presentan los resultados obtenidos de la implementación del prototipo de rastreo de bicicletas con la red Sigfox.

Finalmente, el cuarto capítulo incluye las conclusiones y recomendaciones del trabajo.

PALABRAS CLAVE: Sigfox, IoT, Pycom, Python

ABSTRACT

The present titling work shows a bicycle tracking model with the Sigfox network through the use of Pycom brand nodes. The model consists of 3 main components which are the Sigfox component, the AWS component and the user application; The 3 components will have their design phase, implementation phase and testing phase.

The first chapter contains all the theoretical aspects of the tracking prototype, in which technical aspects of the technologies used in the model are detailed.

The second chapter details the design and implementation of the 3 components: the Sigfox component, the AWS component, and the user application.

The third chapter presents the results obtained from the implementation of the bicycle tracking prototype with the Sigfox network.

Finally, the fourth chapter includes the conclusions and recommendations of the work.

KEYWORDS: Sigfox, IoT, Pycom, Python

1. INTRODUCCIÓN

Es necesario administrar el servicio de prestación de bicicletas en el Distrito Metropolitano de Quito, para optimizar su uso, su tiempo de vida y su ubicación en el momento que están prestando servicio a los usuarios; para esto es necesario contar con un sistema rápido, eficiente y económico que permita solucionar los problemas al usuario cuando una bicicleta falla, realizar un control del uso de las bicicletas, y brindar una asistencia al usuario de forma inmediata entre otras necesidades.

Las técnicas como rastreo satelital o rastreo por medio de los sistemas celulares son soluciones que son factibles implementarlas, pero tienen un alto costo de implementación y operación, lo que desalienta la implementación de una solución para este requisito afectando la calidad de servicio que se da al usuario, y generando perdidas a la empresa que administra el servicio de prestación de bicicletas.

En la actualidad el alto costo del servicio de rastreo GPS está asociado al alto costo para realizar la transferencia de datos desde el nodo sensor de ubicación al computador donde se procesa la información impide gestionar el servicio de prestación de bicicletas, causando que no se puedan dar servicios adicionales que están relacionados con la seguridad de los usuarios, y mantenimiento de las bicicletas. Por lo tanto, la solución que se va a implementar implica la colocación de un nodo sensor de la marca Pycom con GPS y tecnología Sigfox en la bicicleta. Los datos obtenidos del GPS se enviarán a través de la red Sigfox, al servidor de aplicaciones, para que se realice el procesamiento de la información con el fin de gestionar el uso de las bicicletas en el Distrito Metropolitano de Quito.

1.1 OBJETIVOS

1.1.1 OBJETIVO GENERAL

- Implementar un prototipo para el seguimiento de bicicletas y su gestión utilizando la tecnología Sigfox.

1.1.2 OBJETIVOS ESPECÍFICOS

- Estudiar tecnologías LPWAN, enfocado a la tecnología Sigfox.
- Implementar un nodo sensor prototipo con tecnología Sigfox.
- Implementar el sistema de seguimiento de bicicletas.
- Implementar el sistema de comunicaciones con la red de Sigfox.

1.2 ALCANCE

El área de cobertura del seguimiento para bicicletas será limitada al área de cobertura de la red Sigfox en el Distrito Metropolitano de Quito.

El prototipo por implementar utilizará un dispositivo GPS para obtener las coordenadas del nodo y enviará la información mediante la tecnología Sigfox a la nube de Sigfox en el internet. Los datos obtenidos del seguimiento de las bicicletas serán obtenidos del servidor *backend* de la nube Sigfox, diseñado específicamente para enviar los datos al usuario a través del internet.

Es por esta razón que el aplicativo a desarrollar deberá implementar un software de comunicaciones que permitan extraer los datos y un software de procesamiento para la presentación de la información al usuario.

El prototipo de rastreo GPS no permitirá tener un rastreo en tiempo real del dispositivo, teniendo en cuenta las normas de transferencia de datos que rigen a la red Sigfox.

El aplicativo que se desarrollará para la gestión de las bicicletas funcionará en un computador o tableta proporcionando la información requerida por el usuario.

El prototipo constará de un nodo sensor de la marca Pycom; el nodo sensor transmitirá información relacionada a la administración de las bicicletas a la nube de Sigfox; se extraerá la información de la nube de Sigfox por medio de servicios de la nube.

Se implementará el prototipo con el nodo sensor de la marca Pycom (Sipy 1.0) en conjunto con el sensor GPS de la marca Pycom (Pytrack 2.0) que serán colocados en una bicicleta, operando con batería de tipo *LiPo* para este dispositivo. Se pondrá en funcionamiento el aplicativo para visualizar el posicionamiento de la bicicleta, y las funciones que permitan realizar la gestión de estas, la información será presentada en un mapa para su mejor interpretación.

El producto final demostrable para la gestión de las bicicletas es un prototipo que consta del nodo sensor y del servidor en el cual se realiza el procesamiento y presentación de la información, con el cual se verificará la solución planteada.

1.3 MARCO TEÓRICO

1.3.1 PROTOCOLO LPWAN

La tecnología LPWAN ha tenido un avance exponencial dentro del campo del Internet de las cosas (IoT) debido a sus inmejorables prestaciones comparadas con otras tecnologías dentro del campo de las comunicaciones inalámbricas. LPWAN permite tener una amplia cobertura de red, un bajo consumo de batería del dispositivo final y un bajo costo de despliegue y mantenimiento de la red, estas características permiten a LPWAN ser aptas para el manejo de aplicaciones de rastreo, cuidado personal, agricultura de precisión, ciudades inteligentes, entre otras aplicaciones. [1]

LPWAN permite eficiencia en el uso de baterías, una amplia zona de cobertura y un bajo costo de comunicación inalámbrica. Las baterías dentro de los dispositivos finales tienen una duración aproximada de 10 años. El costo de suscripción a redes LPWAN puede ser bajo en relación con las comunicaciones celulares y en algunos casos cuando se cuenta con una propia infraestructura no tiene costo. Algunas tecnologías que utilizan LPWAN (0G) son Sigfox, LoRaWan y NB-IoT. [2]

Tabla 1.1. Tabla comparativa de tecnologías LPWAN *[2]*

	LoRa WAN	**Sigfox**	**NB-IoT**
Modulación	CSS	BPSK	QPSK
Frecuencia	Bandas ISM	Bandas ISM	Bandas LTE
Ancho de Banda	125, 250 y 500 KHz	100 Hz	200 KHz
Velocidad de transmisión	300 bps - 50 kbps	100 bps	200 kbps
Consumo de Energía	Bajo	Bajo	Bajo

LoRaWan es un protocolo desarrollado por la Alianza Lora en el año de 2015 que permite una solución de conectividad de bajo consumo de batería, esta utiliza la modulación Chip Spread Spectrum (CSS) con tres diferentes anchos de banda de 125, 250 y 500 KHz en bandas no licenciadas ISM (Industrial, Scientific and Military) [3]

Narrow Band Internet of Things (NB-IoT) es una tecnología de transmisión estandarizado por 3rd Generation Partnership Project (3GPP) que permite la conectividad de bajo consumo de energía, diseñado especialmente para satisfacer las necesidades del Internet de las Cosas, esta tecnología aprovecha las bandas licenciadas de LTE para realizar la transmisión de sus datos. [2]

En la tabla 1.1 encontramos una comparativa de las 3 tecnologías de LPWAN

1.3.2 TECNOLOGÍA SIGFOX

Sigfox es una compañía francesa fundada por Ludovic Le Moan y Christophe Fourtet que desarrolló una tecnología de solución IoT en el año 2010, esta compañía funciona también como operador de red LPWAN. La tecnología Sigfox se despliega en 70 países y se encuentra en un continuo crecimiento a nivel mundial. En la figura 1.1 podemos apreciar la cobertura de la tecnología Sigfox dentro del territorio continental del Ecuador, la región insular no tiene cobertura de la tecnología Sigfox. [4]

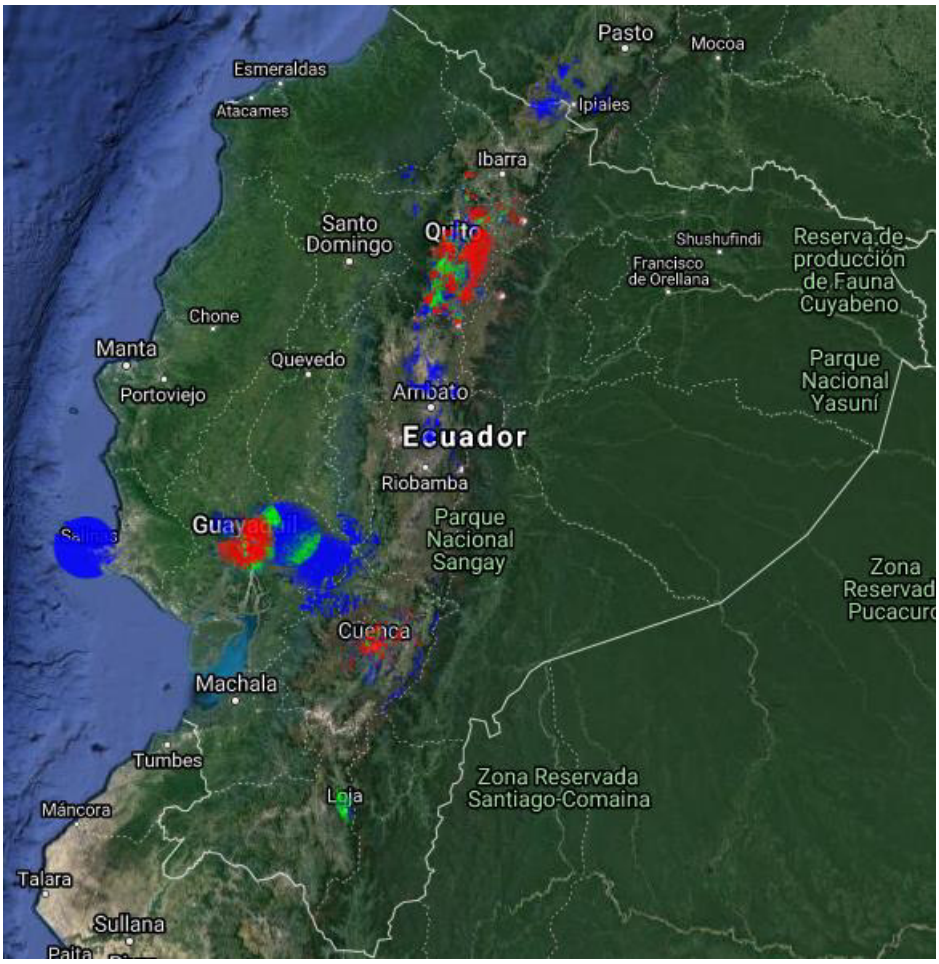


Figura 1.1. Cobertura de Sigfox en Ecuador en el año 2021 *[4]*

Sigfox se encuentra diseñado para cumplir con las características de larga duración de la batería del dispositivo, bajo costo del dispositivo, largo alcance y un bajo costo de conectividad

El protocolo Sigfox en su interfaz de Radio conecta objetos y la red de Sigfox, esta interfaz de radio tiene reglas denominadas 3D-UNB. 3D hace referencia a una triple diversidad (tiempo, espacio y espacio) en bandas estrechas.

Las reglas 3D-UNB están diseñadas para operar en frecuencias no licenciadas, es por lo que Sigfox realiza una clasificación de las configuraciones de Radio acorde a las regulaciones de cada país, esta clasificación se puede apreciar en la figura 1.2. [5]

Frequency (in MHz)	RC1	RC2	RC3	RC4	RC5	RC6	RC7
UL low boundary	868.034	902.104	923.104	920.704	923.204	865.104	868.704
UL center	868.130	902.200	923.200	920.800	923.300	865.200	868.800
UL high boundary	868.226	902.296	923.296	920.896	923.396	865.296	868.896
DL low boundary	869.429	905.104	922.104	922.204	922.204	866.204	869.004
DL center	869.525	905.200	922.200	922.300	922.300	866.300	869.100
DL high boundary	869.621	905.296	922.296	922.396	922.396	866.396	869.196

Figura 1.2. Configuraciones de Radio (RC) definidas por Sigfox *[5]*

En la figura 1.3.2.2 se aprecian canales de 192 KHz de ancho de banda, cuando se requiere de la utilización de canales de menor ancho de banda se tiene la configuración de 6 micro canales de 25 KHz cada uno [5]. El Ecuador se encuentra dentro de la configuración de radio 4.

Las reglas 3D-UNB definen para el enlace *uplink* una pila de procedimientos para el envío de mensajes en el que se define el tamaño del *payload* (0 - 12 bytes), el campo de Identificador compuesto por 4 bytes, el campo de número de mensajes compuesto por 12 bits, el bit REP que se encuentra seteado con el bit 0, el bit BF (*Bidirectional Flag*) y 2 bits para el indicador de longitud de acuerdo con la tabla 1.2 [5]

Tabla 1.2. Valores de LI acorde a otros parámetros del mensaje. *[5]*

Contenido de Payload	LI value (MSB, LSB)		UL-AUTH size (bytes)	UL-Container size (bytes)
Vacío	0	0	2	8
1 byte	0	0	2	9
2 bytes	1	0	4	12
3 bytes	0	1	3	12
4 bytes	0	0	2	12
5 bytes	1	1	5	16
6 bytes	1	0	4	16
7 bytes	0	1	3	16
8 bytes	0	0	2	16
9 bytes	1	1	5	20
10 bytes	1	0	4	20
11 bytes	0	1	3	20
12 bytes	0	0	2	20

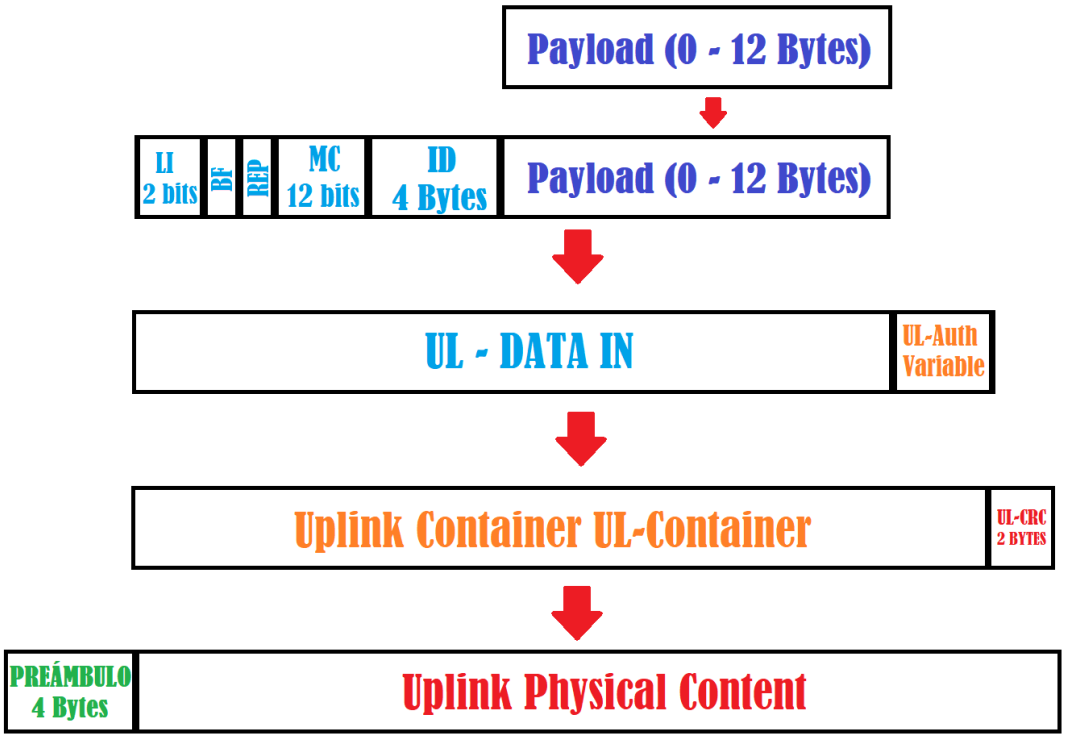


Figura 1.3. Formato de la trama Uplink de Sigfox *[5]*

El campo de autenticación (UL-Auth) se crea con el algoritmo AES128, mediante la concatenación de los campos LI, BF, MC, ID y payload con una clave de autenticación que se genera al momento del registro del dispositivo; este procedimiento genera un resultado de 16 bytes, de los cuales se copian los bytes más significativos, este campo puede tener entre 2 y 5 bytes acorde a la tabla 1.3.

El campo de detección de errores (UL-CRC) se encuentra compuesta por 2 bytes y se la obtiene dividiendo el campo Uplink Container con el polinomio $X^{16} + X^{12} + X^5 + 1$ y el resto se realiza la operación XOR con el valor 0xFFFF.

El campo preámbulo se encuentra compuesto por el campo Tipo de Trama compuesto por 13 bits y el campo Preámbulo compuesto por 19 bits cuyo valor es 0b1010101010101010101; en la figura 1.3 se aprecia todo el procedimiento para la encapsulación del *payload*. [5]

Sigfox utiliza la modulación D-BPSK (*Differential Binary Phase Shift Keying*) en el enlace *uplink*.

El operador autorizado dentro del territorio ecuatoriano es el grupo WND (Wireless Network Development) el cual cuenta con varios niveles de suscripción dependiendo del volumen de datos que serán transmitidos, en la tabla 1.3 encontramos las características de los niveles de suscripción. [4]

Tabla 1.3. Niveles de suscripción de Sigfox *[4]*

Niveles de Suscripción	Mensajes Uplink diarios	Intervalo entre mensajes Uplink	Mensajes Downlink diarios	Intervalo entre mensajes Downlink	Costo por menos de 1000 dispositivos
Platinum	140	10 minutos y 18 segundos	4	6 horas	$5,75
Gold	100	14 minutos y 24 segundos	2	12 horas	$3,75
Silver	50	28 minutos y 48 segundos	1	24 horas	$1,88
One	2	12 horas	0	-	$0,71

El costo de enviar 140 mensajes diarios por medio de la red Sigfox es de $5,75, mientras que al utilizar la mensajería SMS implica un costo por mensaje de $0,112. En el año con la red Sigfox se puede enviar hasta 51.100 mensajes Sigfox; si tuviéramos que utilizar la mensajería SMS tendría un costo anual de $5.723,20. Se puede optar por contratar un plan celular con SMS ilimitados que tiene un costo mensual promedio de $20, que implicaría un costo anual de $240, aunque las operadoras móviles dentro del país tienen una restricción de uso adecuado para estos planes comerciales.

Las reglas 3D-UNB definen la potencia de salida de un dispositivo acorde a su configuración de radio (RC), en el caso de Ecuador la potencia de salida es de 24 dBm. La sensibilidad de las estaciones de radio Sigfox es de -142 dBm, esto permite una perdida en el trayecto de hasta 160 dB. [5]

Figura 1.4. Estructura de la Arquitectura de la red Sigfox [4]

La red Sigfox se encuentra conformada por 3 componentes principales:

- Objetos: Son los dispositivos finales que contienen la tecnología necesaria para poder enviar mensajes a las estaciones base. Este dispositivo final debe tener un identificativo único de 32 bits.
- Estaciones base Sigfox: Son estaciones de radio que son capaces de enviar y receptar mensajes Sigfox a los Objetos por medio de radiofrecuencia con la configuración de radio regulada por cada país, además mantiene comunicación cifrada con la nube de Sigfox.
- Nube de Sigfox: Es la nube propia de Sigfox que permite la recepción, almacenamiento y administración de los mensajes Sigfox enviados por los objetos; cuenta con varios servicios para la integración con otras plataformas IoT.

1.3.3 AMAZON WEB SERVICES (AWS)

Amazon Web Services es una plataforma de Cloud Computing, que ofrece más de 200 servicios integrales de centros de datos a nivel global. Cloud Computing es la distribución de recursos de TI bajo demanda a través de Internet mediante un esquema de pago por uso. En lugar de comprar, poseer y mantener servidores y centros de datos físicos, puede obtener acceso a servicios tecnológicos, como capacidad informática, almacenamiento y bases de datos, en función de sus necesidades a través de un proveedor de la nube como Amazon Web Services (AWS). [6]

AWS cuenta con varios servicios como:

- **Lambda**

AWS Lambda es un servicio informático sin servidor que le permite ejecutar código sin aprovisionar ni administrar servidores, crear una lógica de escalado de clústeres basada en la carga de trabajo, mantener integraciones de eventos o administrar tiempos de ejecución. Con Lambda, puede ejecutar código para casi cualquier tipo de aplicación o servicio backend sin tener que realizar tareas de administración. Los códigos que se pueden ejecutar pueden estar escritos en varios lenguajes como por ejemplo Python. [7]

- **DynamoDB**

Amazon DynamoDB es una base de datos de clave-valor y documentos que ofrece rendimiento en milisegundos de un solo dígito a cualquier escala. Se trata de una base de datos completamente administrada, duradera, multiactiva y de varias regiones que cuenta con copia de seguridad, restauración y seguridad integradas, así como almacenamiento de

caché en memoria para aplicaciones a escala de Internet. DynamoDB puede gestionar más de 10 billones de solicitudes por día y puede admitir picos de más de 20 millones de solicitudes por segundo. [8]

- **Api Gateway**

Amazon API Gateway es un servicio completamente administrado que facilita a los desarrolladores la creación, la publicación, el mantenimiento, el monitoreo y la protección de API a cualquier escala. Las API actúan como la "puerta de entrada" para que las aplicaciones accedan a los datos, la lógica empresarial o la funcionalidad de sus servicios de backend. Con API Gateway, se puede crear API RESTful y API WebSocket que permiten aplicaciones de comunicación bidireccional en tiempo real. API Gateway admite cargas de trabajo en contenedores y sin servidor, así como aplicaciones web. [9]

- **IoT Core**

AWS IoT Core permite conectar dispositivos de IoT a la nube de AWS sin la necesidad de aprovisionar o administrar servidores. AWS IoT Core admite miles de millones de dispositivos y billones de mensajes, y es capaz de procesarlos y direccionarlos a puntos de enlace de AWS y a otros dispositivos de manera confiable y segura. Con AWS IoT Core, las aplicaciones pueden realizar un seguimiento de todos los dispositivos y comunicarse con ellos en todo momento, incluso cuando no están conectados. [10]

- **IAM (Identity and Access Management)**

AWS Identity and Access Management (IAM) puede administrar el acceso a los servicios y recursos de AWS de manera segura. Además, puede crear y administrar usuarios y grupos de AWS, así como utilizar permisos para conceder o negar el acceso de estos a los recursos de AWS. [11]

- **AWS CloudFormation**

CloudFormation permite modelar un conjunto de recursos relacionados de AWS y de terceros, aprovisionarlos de manera rápida y consistente y administrarlos a lo largo de sus ciclos de vida tratando la infraestructura como un código. La plantilla de CloudFormation describe los recursos que desea y sus dependencias para que los pueda lanzar y configurar juntos como una pila. [12]

1.3.4 PYCOM

Pycom es una compañía de desarrollo de Software que fue fundada en el año 2015 en Reino Unido, su nombre comienza del diminutivo del lenguaje de programación Python

“Py” y de la palabra comunicación “com”. La compañía tiene dentro de su catálogo de productos al Pytrack que es una placa de desarrollo que permite obtener la ubicación por medio de su GPS incorporado y además tiene al módulo Sipy que permite la comunicación con la red Sigfox, estos dos componentes cumplen con los requerimientos para poner en práctica el prototipo.

1.3.4.1 PYTRACK 2.0

La placa de desarrollo Pytrack 2.0 tiene varios componentes útiles para la aplicación como el módulo GPS, el conector JST para una batería LiPo, el conector USB para la conexión al computador, los puertos externos de entrada y salida, el cargador de batería LiPo y la ranura para una memoria externa micro SD. [13] Todos los componentes de la placa de desarrollo Pytrack 2.0 se puede apreciar en la figura 1.5.

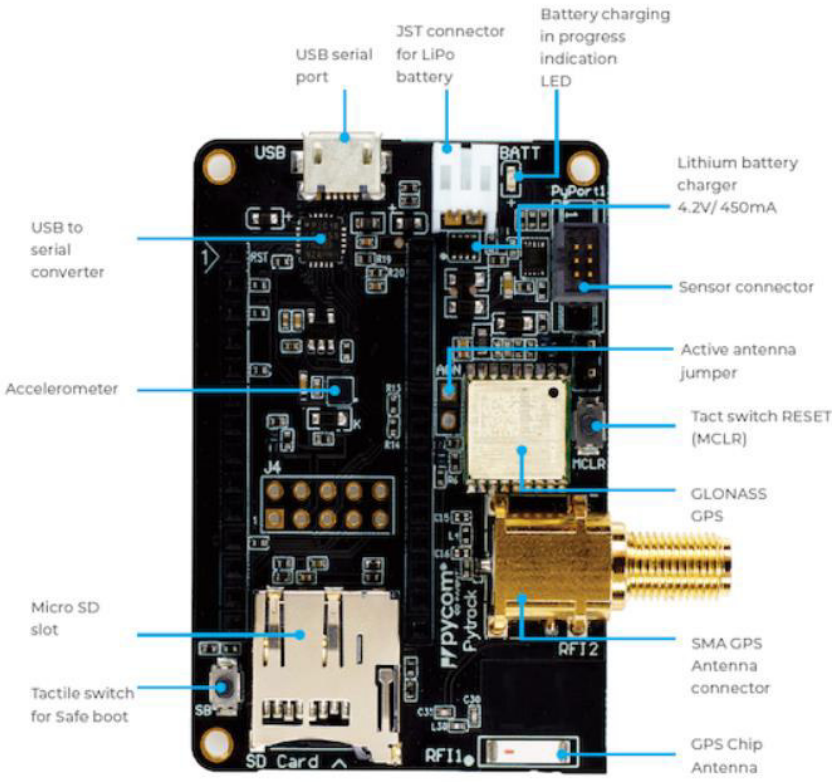


Figura 1.5. Principales componentes de Pytrack 2.0 *[13]*

La placa de desarrollo Pytrack 2.0 permite la integración del módulo Sipy 1.0 para poder configurar el dispositivo por medio del puerto USB y el programa Atom.

1.3.4.2 SIPY

La placa de desarrollo Sipy 1.0 es un conjunto de componentes microcontrolador que opera con el lenguaje de programación MicroPython y contiene 3 tipos de comunicación Sigfox, Wi-Fi y Bluetooth, en la tabla 1.4. podemos apreciar características del módulo.

Tabla 1.4. Características de Sipy 1.0 *[14]*

Características	
Microcontrolador	Xtensa® dual–core 32–bit LX6 microprocessor
Memoria	RAM: 520KB External flash: 4MB
Voltaje de Operación	3.3 V
Wi-Fi	802.11b/g/n
Modelo	0700461242666
Radio Configuration	RC4
Sigfox Uplink Frecuency	920800 KHz

El módulo Sipy 1.0 permite la comunicación con la red Sigfox por medio de su chip CC1125 de la marca Texas Instruments. [14]

El módulo Sipy 1.0 debe ser activado en la red Sigfox antes de su uso, para esto se debe utilizar el programa “Pycom Firmware Update” para obtener la información del Sigfox ID y Sigfox PAC, el proceso de activación del nodo lo podemos obtener en el Anexo 1 .

1.3.5 ATOM

Atom es un software abierto que permite la edición de código en varios lenguajes de programación y por medio de sus complementos se puede instalar el software Pymakr que permite enlazar los dispositivos Pycom con el principal objetivo de cargar el código a los dispositivos.

1.3.6 GNSS (GALILEO SATELLITE NAVIGATION SYSTEM)

El sistema de navegación por satélite Galileo es construido y operado por la UE. En 1999, para satisfacer la creciente demanda de servicios posicionamiento y reducir la dependencia del GPS de EE. UU y del GLONASS de Rusia, la UE decidió construir un sistema propio,

denominado Galileo. A diferencia de GPS y GLONASS, Galileo se construyó originalmente como un sistema de navegación abierto para usuarios civiles de navegación global. [15]

GNSS tiene actualmente 22 satélites operativos y 4 que no se encuentran disponibles y no contribuyen al sistema. [16]

1.3.7 REST

REST (Representational State Transfer) es un tipo de arquitectura de software que fue diseñada para garantizar la interoperabilidad entre diferentes sistemas informáticos de Internet, este aprovecha las capacidades del protocolo HTTP (Hypertext Transfer Protocol) y el URI (Uniform Resource Identifier) para obtener o modificar un recurso. [17]

Se realiza peticiones al servidor mediante el protocolo HTTP usando los métodos GET, POST, PUT y DELETE. Una vez que se recibe la solicitud, las API diseñadas para REST (conocidas como API o servicios web de RESTful) pueden devolver mensajes en distintos formatos: HTML, XML, texto sin formato y JSON. [18]

1.3.7.1 API REST

Las aplicaciones hacen uso de las API (Application Programming Interface) para poder hacer uso de recursos. La aplicación realiza intercambio de mensajes con el servidor. El servicio de API Gateway permite el servicio de API Rest con seguridades y se puede hacer uso de una herramienta computacional sin un servidor dedicado como lo es Lambda.

1.3.7.2 API CALLBACK

La API Callback permite responder a eventos que ocurren y realizar acciones que se requieran para un fin específico. En el caso de la nube sigfox se permite la configuración de la API Callback para el reenvío de los mensajes Sigfox a la plataforma de nuestra preferencia.

1.3.8 PYTHON

Python es un lenguaje de programación interpretado, orientado a objetos y de alto nivel con semántica dinámica. Sus estructuras de datos integradas de alto nivel, combinadas con tipado dinámico y enlace dinámico, lo hacen muy atractivo para el desarrollo rápido de aplicaciones, así como para su uso como lenguaje de scripts o pegamento para conectar componentes existentes. [19]

1.3.8.1 MICROPYTHON

MicroPython es una implementación sencilla y eficiente del lenguaje de programación Python 3 que incluye un pequeño subconjunto de la biblioteca estándar de Python y está optimizado para ejecutarse en microcontroladores y en entornos restringidos.

MicroPython está repleto de características avanzadas como un mensaje interactivo, enteros de precisión arbitraria, cierres, comprensión de listas, generadores, manejo de excepciones y más. Sin embargo, es lo suficientemente compacto como para caber y ejecutarse en solo 256k de espacio de código y 16k de RAM. [20]

1.3.9 JAVASCRIPT

JavaScript es un lenguaje de programación interpretado, por lo que no es necesario compilar los programas para ejecutarlos. En otras palabras, los programas escritos con JavaScript se pueden probar directamente en cualquier navegador sin necesidad de procesos intermedios. [21]

1.3.9.1 REACTJS

ReactJS es una biblioteca de JavaScript que se implementa para desarrollar componentes de interfaz de usuario (UI) reutilizables. ReactJs permite el desarrollo de aplicaciones web grandes y complejas que pueden cambiar sus datos sin actualizaciones de página posteriores.

2. METODOLOGÍA

2.1 FASE DE DISEÑO

2.1.1 REQUISITOS DEL PROTOTIPO

El prototipo de rastreo de bicicletas necesita de varios componentes para su puesta en práctica, entre los principales componentes tenemos un sistema de navegación satelital que proporcione información acerca de la ubicación, un sistema de comunicación de la información de la ubicación de bajo costo y que tenga una gran cobertura, un sistema de almacenamiento y decodificación de mensajes y un sistema que permita el acceso a las bases de datos para la aplicación. Un diagrama general del prototipo con todos sus componentes se encuentra en la figura 2.1.

El componente que permita obtener la información de la ubicación de la bicicleta será la red de satélites GNSS (Galileo Satellite Navigation System).

El componente que permita una comunicación de la ubicación de la bicicleta es la red Sigfox, debido a sus características de bajo costo y amplia cobertura.

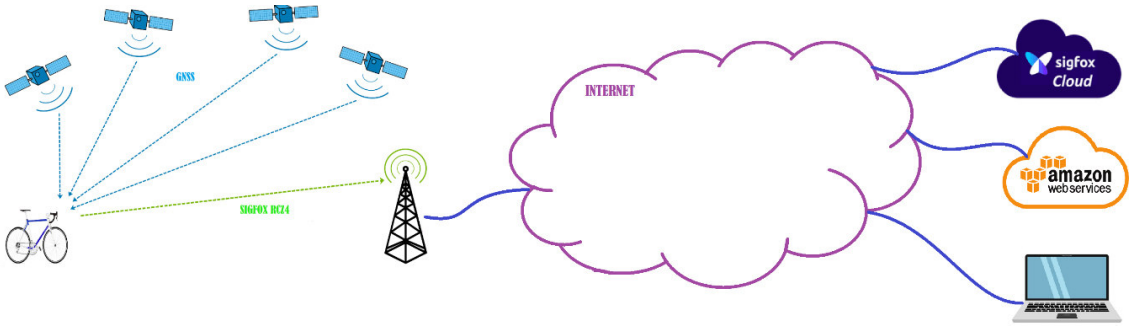


Figura 2.1 Esquema general del prototipo de Hardware.

El componente encargado de la decodificación del mensaje y el almacenamiento de la información de la ubicación de la bicicleta y permitir el acceso a las bases de datos para la aplicación será la nube AWS (Amazon Web Services).

El componente de aplicación de usuario se encargará de interactuar adecuadamente a las bases de datos de la nube AWS y presentar la información requerida al usuario final.

Los componentes principales del prototipo se encuentran especificados en la figura 2.2 cuyos componentes se encuentran separados 3 partes que son el componente Sigfox, el componente AWS y la aplicación de usuario.

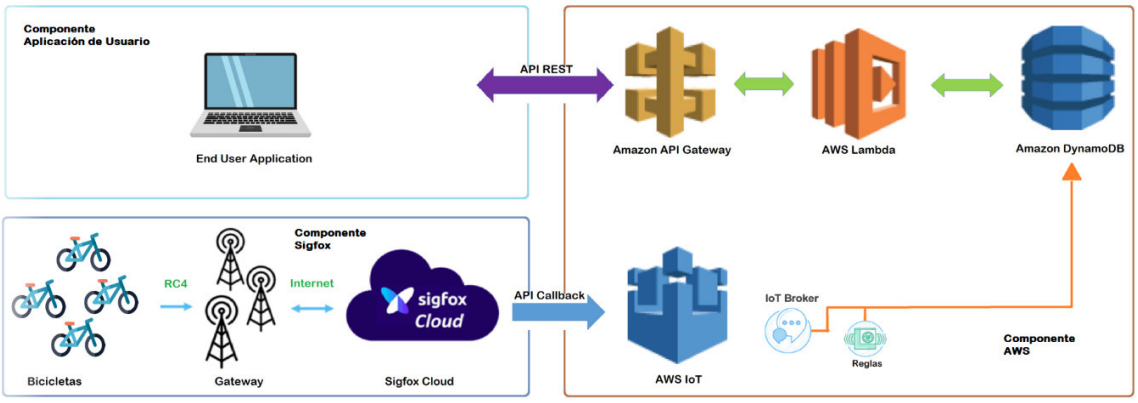


Figura 2.2 Esquema general del prototipo de Software.

2.1.2 DISEÑO DE COMPONENTE SIGFOX

El componente Sigfox se encuentra conformado por el nodo Sigfox, las estaciones de base Sigfox y la nube de Sigfox. Los componentes de estaciones de base Sigfox y la nube de Sigfox son implementados por el operador autorizado en el país, por lo que no se realiza la implementación de estos componentes solo se hacen uso de estos.

2.1.2.1 Nodo Sigfox

El nodo Sigfox se encuentra conformado por una bicicleta y un módulo Sigfox que permita enviar la ubicación a la red Sigfox.

El dispositivo que se adapta a este requerimiento es el conjunto de placas de desarrollo de la marca Pycom Sipy 1.0 y Pytrack 2.0; el conjunto de dispositivos permite obtener la información de la ubicación por medio de la red de satélites GNSS y enviar la información de la ubicación por medio de la red Sigfox en la configuración de radio 4 para el país de Ecuador.

El módulo Sigfox cuenta con una batería LiPo de 3.7 V, una antena externa Sigfox, un botón de emergencia conectado al pin externo 8 con conexión Pull Up, el módulo Sipy 1.0 y el módulo Pytrack 2.0; en la figura 2.3 se puede apreciar un modelo del módulo Sigfox.

Figura 2.3 Módulo Sigfox.

2.1.2.2 Diseño de código del Nodo Sigfox

El conjunto de dispositivos Pycom deben permitir la ubicación del módulo, el envío de mensajes Sigfox y la implementación de un botón de emergencia; para estos propósitos se utilizan librerías propias del módulo Pycom. El diagrama de flujo del código implementado se puede apreciar en la figura 2.4 Para obtener la ubicación de la bicicleta se utiliza la red de satélites GNSS de la Unión Europea por medio de la antena incorporada en la placa de desarrollo Pytrack 2.0, si se requiere de mayor cobertura se tiene la posibilidad de utilizar una antena externa. El código debe permitir enviar mensajes con la ubicación y el estado del usuario cada 11 minutos, si se tiene una emergencia se debe tener la posibilidad de enviar un mensaje con la ubicación y el estado de alarma cuando se presione el botón de emergencia.

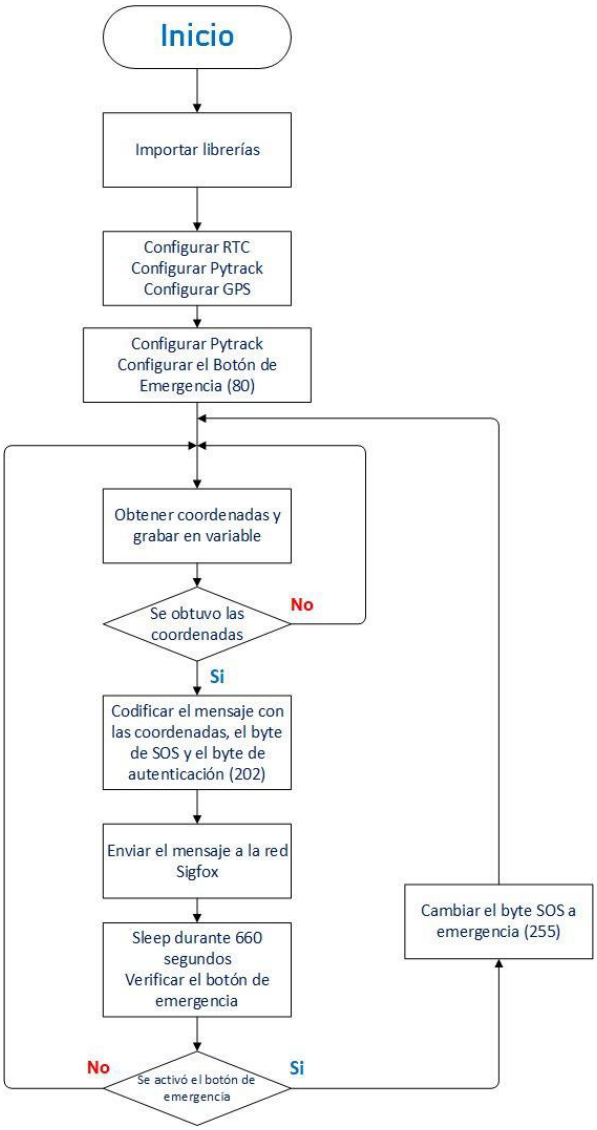


Figura 2.4 Diagrama de flujo Nodo Pycom.

El *payload* de enlace de subida es de 12 bytes en el que se transporta la información correspondiente a la ubicación en coordenadas geográficas (latitud y longitud), la información del botón de emergencia y la autenticación del *payload*.

En la figura 2.5 se puede apreciar la distribución de los 12 bytes del *payload* para el prototipo.

BYTE 1	BYTE 2	BYTE 3	BYTE 4	BYTE 5	BYTE 6	BYTE 7	BYTE 8	BYTE 9	BYTE 10	BYTE 11	BYTE 12
Latitud norte (0) Latitud sur (1)	Latitud valor entero entre 0 y 90	Latitud valor décima y centésima entre 0 y 99	Latitud valor milésima y diez milésima entre 0 y 99	Latitud valor cien milésima y millonésima entre 0 y 99	Byte de Auth del Payload (0x CA)	Longitud este (0) Longitud oeste (1)	Longitud valor entero entre 0 y 180	Longitud valor décima y centésima entre 0 y 99	Longitud valor milésima y diez milésima entre 0 y 99	Longitud valor cien milésima y millonésima entre 0 y 99	Byte SOS 80 Normal 255 Emergencia

Figura 2.5 Payload del Nodo Sigfox.

La latitud es un valor que proporciona la ubicación al norte o sur del Ecuador, mientras que la longitud es un valor que proporciona la ubicación al este u oeste del meridiano de Greenwich. El rango de valores de la latitud es entre -90 y 90 y el rango de valores de la longitud es entre -180 y 180.

En la figura 2.5 podemos apreciar el esquema de distribución de los 12 bytes de *payload* de Sigfox, por ejemplo, si tenemos los valores de latitud igual a -0.211602, de longitud igual a -78.490357 y sin emergencia se tendrá el siguiente esquema

Byte 1: 0x01 Byte 2: 0x00

Byte 3: 0x15 Byte 4: 0x10

Byte 5: 0x02 Byte 6: 0xCA

Byte 7: 0x01 Byte 8: 0x4E

Byte 9: 0x31 Byte 10: 0x03

Byte 11: 0x39 Byte 12: 0x50

2.1.2.3 Estaciones Base Sigfox

Las estaciones base mantiene comunicación con los Nodos Sigfox que se encuentran en las bicicletas, para el prototipo requerimos que solo se tenga comunicación *uplink* desde el nodo hacia la nube Sigfox, la cobertura de las estaciones de base debe cubrir el Distrito Metropolitano de Quito con al menos 1 estación. En la figura 2.6 apreciamos la cobertura Sigfox en el Distrito Metropolitano de Quito en el cual con el color rojo se tiene cobertura

con 3 estaciones base, con el color verde se tiene cobertura con 2 estaciones base, el color azul se tiene cobertura con 1 estación base, los lugares donde no se tiene cobertura no tiene vinculado ningún color.

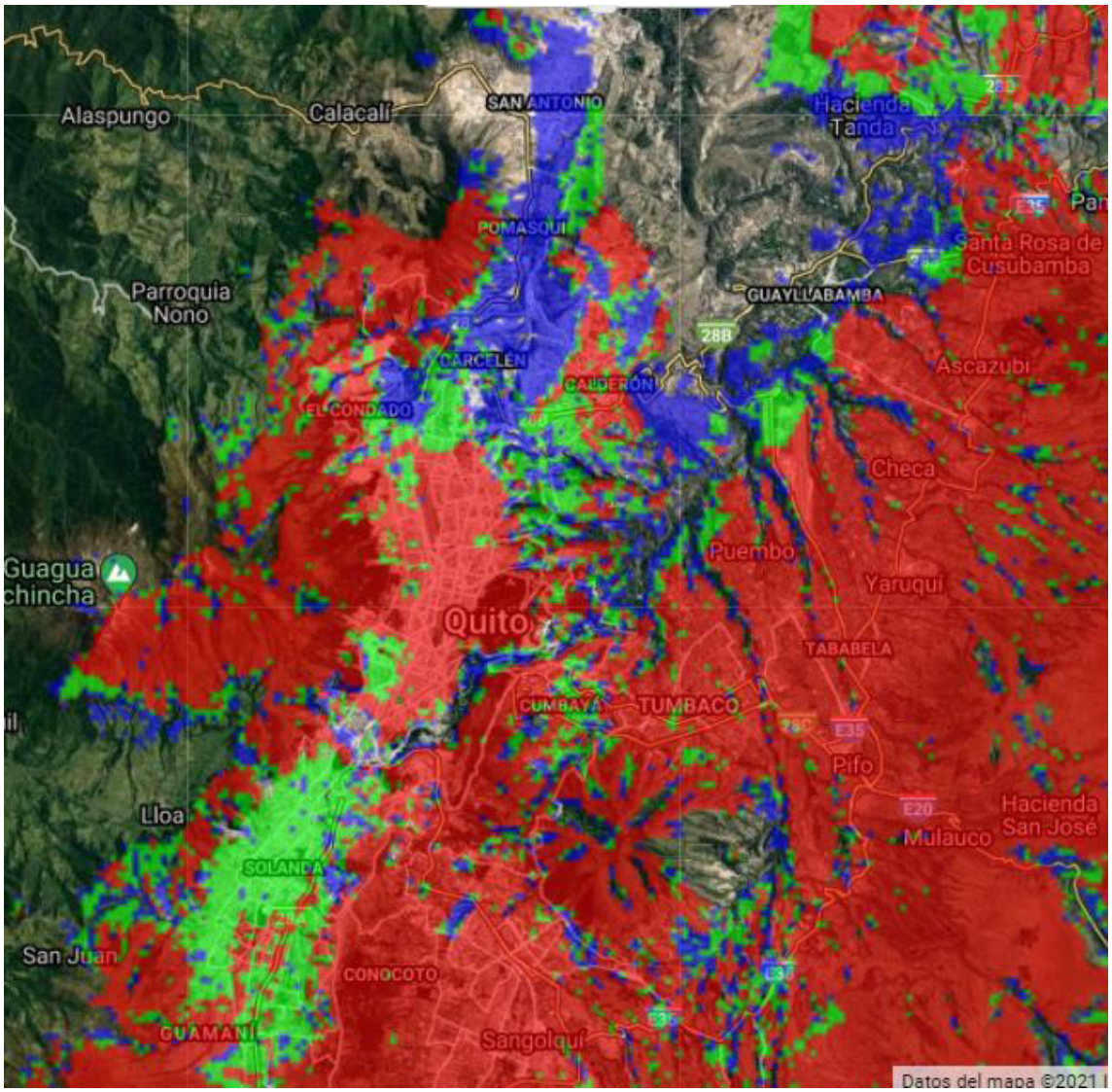


Figura 2.6. Cobertura Sigfox en el Distrito Metropolitano de Quito en 2021. *[4]*

2.1.2.4 Nube Sigfox

Una vez transmitido el mensaje de Sigfox por medio del nodo Sigfox, este mensaje llega a la nube de Sigfox en el que se realiza el almacenamiento y administración de los mensajes, además permite la integración de nuestras propias aplicaciones por medio de API (Application Programming Interfaces).

La nube de Sigfox permite 2 tipos de API que son API Rest y API Callback; la API Callback es la API que se utiliza en el prototipo para la comunicación entre la Nube de Sigfox y la Nube de Amazon. El acceso a la nube de Sigfox se lo realiza por medio del link

https://backend.sigfox.com/, en el anexo 1 se tiene un instructivo de registro de dispositivos en la nube de Sigfox.

Para el acceso a la plataforma de Sigfox se debe crear un usuario y registrar los dispositivos que se van a utilizar en el prototipo y seleccionar el nivel de suscripción deseado.

2.1.3 DISEÑO DE COMPONENTE AWS

El componente AWS, que se encuentra en la figura 2.2, se encarga de decodificar los mensajes Sigfox, proporcionar un método de acceso a las bases de datos y almacenar la información proporcionada por los nodos Sigfox y el aplicativo de usuario.

El componente AWS se encuentra conformado por varios servicios propios de la nube AWS como el gestor de acceso IAM, el IoT Core, la base de datos DynamoDB y el servicio de API Gateway.

2.1.3.1 AWS IAM (Identity Access Management).

El servicio de Amazon IAM permitirá la creación de identidades que tendrán roles específicos dentro del funcionamiento de la aplicación, estos roles adquieren permisos especiales para poder ejecutar funciones dentro de la aplicación, para el prototipo se crea un rol con todos los permisos necesarios para acceder a todas las funcionalidades.

2.1.3.2 AWS Cloud Formatio

El servicio de Cloud Formation permitirá crear el vínculo entre la nube de Sigfox y los servicios de AWS, para lograr este propósito se crea un script que contenga los parámetros necesarios para vincular ambos servicios. El script base para crear la API Callback ya se encuentra creada y se la puede obtener del link:

https://s3-eu-west-1.amazonaws.com/cloud-formation-script/Sigfox_Cross_Account_Access_Role_AWS_IoT_v2.json

2.1.3.3 AWS IoT Core

El servicio de AWS IoT Core permitirá el monitoreo de los mensajes Sigfox que ingrese por medio de la API Callback, también permite crear reglas para ejecutar las funciones de Lambda para almacenar, filtrar y decodificar los mensajes que son requeridos para la aplicación.

2.1.3.3.1 regla AWS IoT core

La regla de AWS IoT Core permitirá que todos los mensajes que lleguen a la plataforma IoT Core se almacenen en una tabla del servicio de base de datos DynamoDB y también permitirá filtrar, decodificar y almacenar en una tabla del servicio de base de datos

DynamoDB los mensajes con los elementos identificación del dispositivo, marca de tiempo, indicador de emergencia, latitud y longitud, también se almacenará el último valor de latitud, longitud, SOS en otra tabla para poder mostrar la ubicación de todos los dispositivos. La última parte de la regla deberá ser ejecutada con una herramienta computacional sin servidor debido a su complejidad.

El diagrama de flujo de la aplicación de Lambda se aprecia en la figura 2.7.

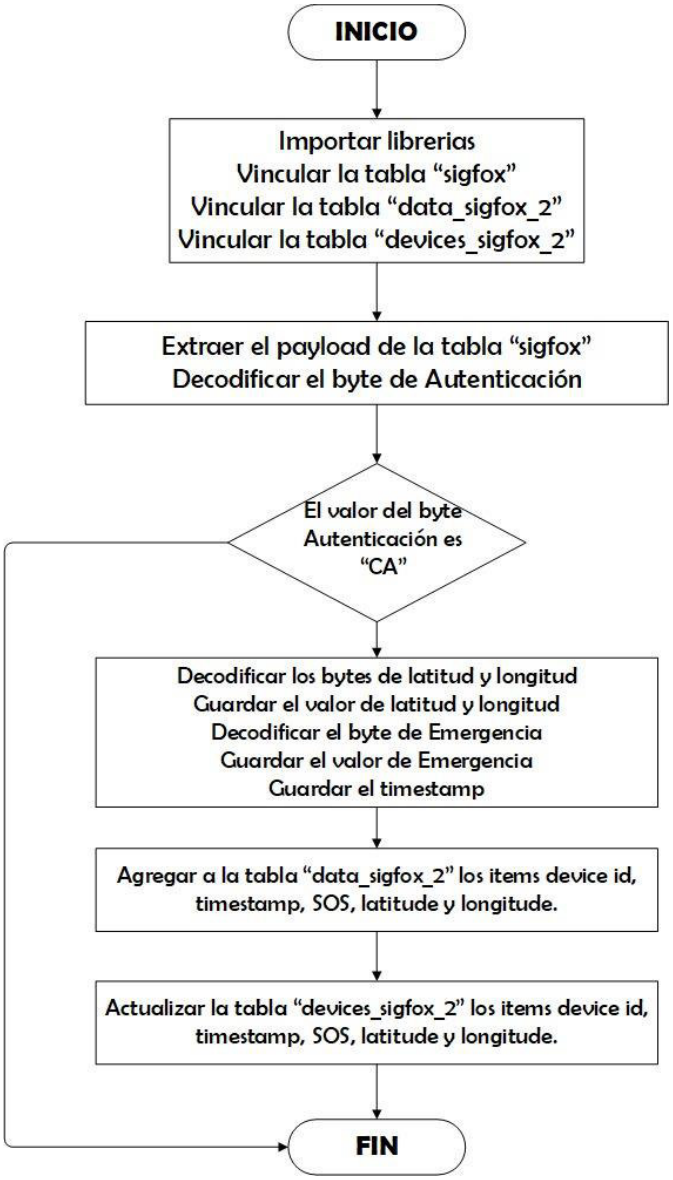


Figura 2.7. Diagrama de flujo de la Regla de IoT core.

2.1.3.4 AWS DynamoDB

El servicio de AWS DynamoDB permite el manejo de bases de datos que permitirá el almacenamiento de la información de los mensajes de la ubicación de varias bicicletas y

además se utilizará este servicio para crear una base de datos de usuarios y registros de uso de la aplicación. En la tabla 2.1 se puede apreciar las tablas creadas y sus funcionalidades.

Tabla 2.1. Funciones de las tablas DynamoDB.

Tablas DynamoDB	Función
data_sigfox_2	Se encarga de almacenar la información decodificada del payload Sigfox para la aplicación.
devices_sigfox_2	Se encarga de llevar un registro de los dispositivos que se encuentran vinculados con la aplicación y almacena la última ubicación y estado de emergencia de cada dispositivo.
historial_sigfox	Se encarga de llevar un registro de uso de los dispositivos según el usuario y dispositivo ocupado.
sigfox	Se encarga de almacenar todos los mensajes provenientes de la nube Sigfox.
usuarios_sigfox	Se encarga de llevar un registro de todos los usuarios registrados en la aplicación y sus datos personales.

2.1.3.5 AWS Api Gateway

Se requiere de un Api que permita la comunicación entre las tablas de DynamoDB y la aplicación de usuario es por lo que el servicio de Api Gateway de AWS permitirá la comunicación de las bases de datos con la aplicación, la API Rest será utilizada para la comunicación entre estos dos componentes. Para el uso de este servicio es necesario de dotarle de una herramienta computacional que será otorgado por el servicio de AWS Lambda y para facilidad se crea todas las APIs con el método GET y sin claves de seguridad. En la tabla 2.2 se puede apreciar las APIs requeridas con su funcionalidad.

Tabla 2.2. Funciones de las APIs.

APIs	Función
TransactionApis_Get_Devices_Info	Permite consultar la información de la tabla devices_sigfox_2.
TransactionApis_Get_History_Info	Permite obtener la información del historial de uso del aplicativo por medio del número de cédula de la tabla historial_sigfox.
TransactionApis_Modify_Status_from_Devices	Permite cambiar el estado de las bicicletas a Libre o Ocupado según se requiera de la tabla devices_sigfox_2.

TransactionApis_Modify_Status_from_Users	Permite consultar la información de la tabla usuarios_sigfox.
TransactionApis_ModifyStatus	Permite cambiar el Id y status del usuario de la tabla usuarios_sigfox.
TransactionApis_ModifyUser	Permite modificar datos personales del usuario de la tabla usuarios_sigfox
TransactionApis_NewUser	Permite crear un nuevo usuario y valida si el usuario ya se encuentra creado en la tabla usuarios_sigfox.

2.1.3.6 AWS Lambda

El servicio de AWS Lambda es la herramienta computacional que permitirá la implementación de código sin necesidad de tener un servidor, esta herramienta será utilizada con el lenguaje de programación Python para filtrar, decodificar y almacenar los mensajes provenientes de la nube Sigfox; también se utiliza esta herramienta para crear y modificar usuarios en la base de datos DynamoDB.

2.1.4 DISEÑO DE COMPONENTE APLICACIÓN.

El componente Aplicación, que se encuentra en la figura 2.2, se encarga de brindar un interfaz al usuario final para que pueda utilizar el prototipo de rastreo de bicicletas.

La aplicación constará de un servidor web Node JS en el que se desarrolla la aplicación con React JS en el editor de código Visual Studio Code. La aplicación de usuario utiliza las tablas generadas en Amazon Web Service en conjunto con la Api de Google Maps para poder mostrar de forma atractiva la ubicación de las bicicletas a los usuarios.

Figura 2.8. Software de la Aplicación de Usuario.

La aplicación funcionará dentro del computador por medio del servidor web y podrá ser utilizado en otros dispositivos por medio de una red local.

La aplicación debe permitir las siguientes actividades dentro de su diseño:

- Crear nuevos usuarios.
- Iniciar sesión.

- Reservar una bicicleta.
- Observar el historial del usuario.
- Quitar la reserva de una bicicleta.

En la figura 2.9 se puede apreciar el diagrama de máquina de estados que especifica la secuencia de eventos dentro del aplicativo.

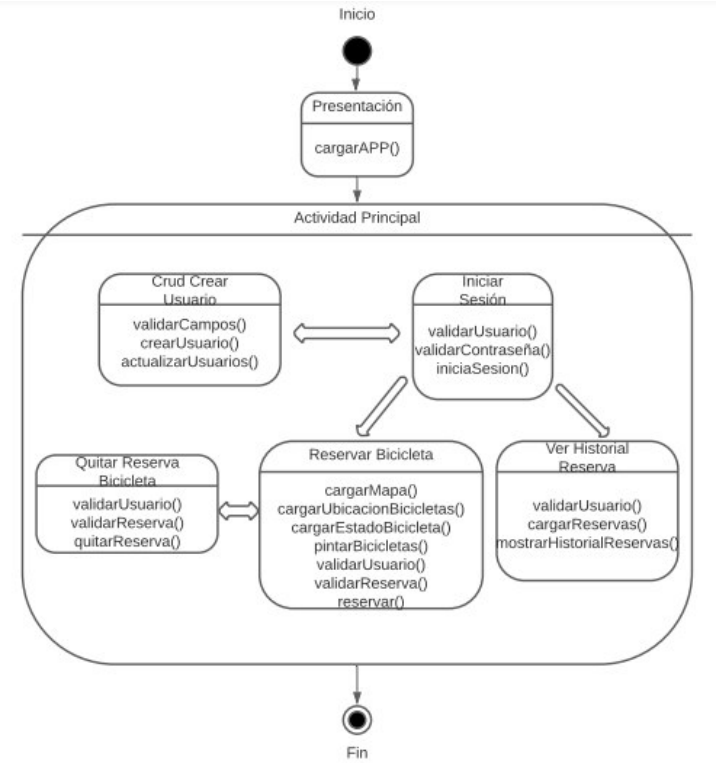


Figura 2.9. Diagrama de máquina de estados de la aplicación.

2.1.4.1 Crear Usuario

En la aplicación es necesario contar con un mecanismo para crear los usuarios que harán uso de la aplicación, para esto es necesario crear un formulario para ingresar los datos del nuevo usuario y crearlo en la tabla de *usuarios_sigfox*; para esto es necesario validar los datos dentro del formulario para no tener datos erróneos dentro de la tabla.

Los datos personales que deberán proporcionar los nuevos usuarios son los siguientes:

- Cédula de identidad
- Correo electrónico
- Número de celular
- Nombres completos
- Número de contacto
- Contraseña

2.1.4.2 Iniciar Sesión

La aplicación permitirá el inicio de sesión de los usuarios registrados previamente, para esto se validarán los datos proporcionados en el formulario de inicio de sesión; el usuario para poder ingresar a la aplicación tendrá que ingresar el número de cédula y la contraseña, si los valores coinciden con los registrados en la tabla *usuarios_sigfox* se valida el ingreso, caso contrario se muestra un mensaje de error.

2.1.4.3 Reservar Bicicleta

La aplicación permitirá la reserva de bicicleta que se encuentren disponibles al momento de realizar la reserva, para esto se hace uso del api de Google maps para mostrar la información de la ubicación de las bicicletas disponibles y su última ubicación. Al momento de la reserva se obtendrá información de la bicicleta como el ID de Sigfox y se cambiará el estado de la bicicleta reservada a ocupada para que no se muestre disponible, también se empieza el registro del historial del usuario.

2.1.4.3.1 api de Google maps

La aplicación requiere mostrar la ubicación de las bicicletas en un mapa, para lo cual se hace uso del api de Google maps, para lo cual se requiere crear un usuario en la nube de Google y solicitar la clave de acceso para uso del api de Google maps.

2.1.4.4 Quitar reserva Bicicleta

La aplicación permitirá terminar la reserva de una bicicleta que se ha reservado previamente, al momento de realizar este procedimiento se da por terminado el historial del usuario con la bicicleta que se encontraba usando y la bicicleta se cambia a estado de disponible.

2.1.4.5 Historial Bicicleta

La aplicación permitirá visualizar en una tabla y en un mapa el historial de ubicaciones del usuario de las últimas 24 horas o sus 15 últimas ubicaciones.

2.2 FASE DE IMPLEMENTACIÓN

2.2.1 CÓDIGO NODO SIGFOX

El diagrama de flujo de la figura 2.7 describe las funcionalidades del prototipo; el código implementado está escrito en el lenguaje de programación MicroPython y se utiliza librerías de la marca Pycom con Licencia Pública General GNU (GNU GPL o GPL), que es una licencia de software libre ampliamente utilizada, que garantiza a los usuarios finales la libertad de ejecutar, estudiar, compartir y modificar el software.

2.2.1.1 Librería L76GNSS

La librería L76GNSS de propiedad de la marca Pycom permite obtener la latitud y longitud por medio del módulo Pytrack 2.0 con su antena integrada o su antena externa.

2.2.1.2 Librería Pycropoc

La librería pycropoc permite vincular los módulos Sipy 1.0 y el Pytrack 2.0 para el intercambio de información, por ejemplo, los datos de la ubicación, el nivel de voltaje de la batería, el estado de los pines externos, entre otros.

2.2.1.3 Función de Codificación del mensaje

La función de codificación del mensaje permite crear los 5 bytes de longitud y los 5 bytes de latitud de la aplicación acorde al diseño de la figura 2.5 El código de la función de codificación del mensaje se aprecia en la figura 2.10.

En la función se utiliza 10 variables para identificar los 5 bytes de la longitud y los 5 bytes de latitud, en el que se guardan los valores enteros y decimales de la ubicación, además cuenta con una variable para identificar si el valor es positivo o negativo; el código espera que se obtenga la ubicación del sensor GPS para proceder con la codificación, en el caso de no obtener la ubicación se coloca el led en color rojo y procede a obtener la ubicación del sensor otra vez.

La función codificación devuelve 10 variables en el caso de haber obtenido la ubicación del sensor y en el caso de no obtener la ubicación devuelve el valor de 10.

```
def Codificacion(a):
    negativo=-1
    positivo=1
    latitud=a[0]
    longitud=a[1]
    if latitud is None:
        byte1Lat=255
        byte1Lon=255
        print('Obteniendo ubicacion GPS...')
        time.sleep(1)
        pycom.rgbled(0x090000) #Led en color rojo
        return 10
    else:
        pycom.rgbled(0x000001) #Led en color azul
        print(latitud)
        print(longitud)
        if latitud<=0:
            byte1Lat=1
            byte1Lat=int(byte1Lat)
            latitud=math.copysign(latitud, positivo)
        else:
            byte1Lat=0
            byte1Lat=int(byte1Lat)
        if longitud<=0:
            byte1Lon=1
            byte1Lon=int(byte1Lon)
            longitud=math.copysign(longitud, positivo)
        else:
            byte1Lon=0
            byte1Lon=int(byte1Lon)
        latitud_entero=math.floor(latitud)
        byte2Lat=int(latitud_entero)
        longitud_entero=math.floor(longitud)
        byte2Lon=int(longitud_entero)
        latitud_frac=(latitud-latitud_entero)*1000000
        longitud_frac=(longitud-longitud_entero)*1000000
        byte345Lat=int(math.floor(latitud_frac))
        byte91011Lon=int(math.floor(longitud_frac))

        byte9=(int(math.floor(byte91011Lon/10000)))
        byte10=byte91011Lon-byte9*10000
        byte10=(int(math.floor(byte10/100)))
        byte11=byte91011Lon-byte9*10000-byte10*100
        byte11=(int(byte11))

        byte3=(int(math.floor(byte345Lat/10000)))
        byte4=byte345Lat-byte3*10000
        byte4=(int(math.floor(byte4/100)))
        byte5=byte345Lat-byte3*10000-byte4*100
        byte5=(int(byte5))

        return [byte1Lat, byte2Lat, byte3, byte4,byte5, byte1Lon,
byte2Lon, byte9, byte10, byte11]
```

Figura 2.10 Función codificación del mensaje.

2.2.1.4 Función de envío de mensaje Sigfox

La función de envío de mensaje Sigfox permite el envío de los 12 bytes establecidos en la figura 2.5 y configura la red Sigfox en con el radio de configuración 4 especificado para Ecuador. El código de la función envío de mensaje se aprecia en la figura 2.11.

En la función requiere del envío de los 11 bytes para poder organizarlos acorde la figura 2.1.2.2.2 y agrega el byte de autenticación de la aplicación con el valor de 0xCA.

```
def EnviarDatoSigfox(byte1Lat, byte2Lat, byte3, byte4,byte5,
byte1Lon, byte2Lon, byte9, byte10, byte11, SoS):
    from network import Sigfox
    import socket
    sigfox=Sigfox(mode=Sigfox.SIGFOX, rcz=Sigfox.RCZ4) #RCZ4 ->
Ecuador
    s=socket.socket(socket.AF_SIGFOX, socket.SOCK_RAW)
    s.setblocking(True)
    s.setsockopt(socket.SOL_SIGFOX, socket.SO_RX, False)
    pycom.rgbled(0x010001) #Led Rosado
    s.send(bytes([byte1Lat, byte2Lat, byte3, byte4,byte5, 0xCA,
byte1Lon, byte2Lon, byte9, byte10, byte11, SoS]))
    pycom.rgbled(0x0F0F0F) #Led Blanco
    print('!!Mensaje enviado!!')
```

Figura 2.11 Código para envío de 12 bytes a la red Sigfox.

2.2.1.5 Código Nodo Sigfox

El código principal del nodo Sigfox sigue el diagrama de flujo de la figura 2.5 en el que se encarga de vincular las funciones y librerías antes mencionadas con el objetivo de enviar la latitud, longitud, estado de emergencia y el byte de tipo de mensaje para poder identificar la aplicación.

2.2.1.5.1 importar librerías necesarias

Para el código principal del nodo Sigfox es necesario importar varias librerías que son utilizadas en el código, entre las principales librerías se encuentra la librería Pycropoc y la librería L76GNSS. En la figura 2.12 se aprecia el extracto del código principal en el que se importan las librerías.

```
import machine
import math
import network
import os
import time
import utime
import gc
import pycom
import binascii
from machine import RTC
from machine import Pin
from L76GNSS import L76GNSS
from pycoproc_2 import Pycoproc
```

Figura 2.12. Código para importar librerías.

2.2.1.5.2 configuraciones iniciales

Para el código principal del nodo Sigfox es necesario inicializar las variables con las que se manejara las funcionalidades del código y pasar los parámetros iniciales a las funciones detalladas anteriormente. En la figura 2.13 se aprecia el código principal en el que se realiza las configuraciones iniciales.

```
pycom.heartbeat(False)
pycom.rgbled(0x030301) #Led Blanco
time.sleep(2)
gc.enable()
rtc = machine.RTC()
rtc.ntp_sync("pool.ntp.org")
utime.sleep_ms(750)
print('\nRTC Set from NTP to UTC:', rtc.now())
utime.timezone(7200)
print('Adjusted from UTC to EST timezone', utime.localtime(), '\n')
py=Pycoproc() # configurar Pytrack2.0
time.sleep(1)
l76=L76GNSS(py, timeout=30, buffer=512)
pybytes_enabled = False
emergency=0
```

Figura 2.13. Código de configuraciones iniciales.

2.2.1.5.3 lazo while principal

Para el código principal del nodo Sigfox es necesario crear un lazo while para que se ejecute el código todo el tiempo hasta que se produzca una interrupción externa ocasionada por falta de energía o por reiniciar el nodo Sigfox. En el código se obtiene la ubicación por medio del sensor GPS y se envía a codificar la información obtenida, en el caso de no obtener la ubicación se crea un tiempo de espera de 5 segundos para volver a pedir la ubicación al sensor caso contrario se envía la información codificada a la red Sigfox

con la función “EnviarDatoSigfox” y se crea un lazo while por el tiempo de espera requerido para volver a enviar la información de la ubicación en el que se realiza un censado del pin para ver si se produce el evento de emergencia, si se mantiene presionado el botón por 5 segundos el valor de la variable SoS cambia por el valor decimal de 255 y se envía ese momento la ubicación del nodo Sigfox.

En la figura 2.14 se aprecia el extracto del código principal en el que se encuentra el lazo while principal del nodo Sigfox.

```
while (True):
    coord=l76.coordinates() #Obtener ubicacion
    a=list(coord)
    data=Codificacion(a)#Obtener datos de latitud y longitud
    if data == 10:#Condicional si no se obtuvo ubicacion
        print('No data')
        time.sleep(5)
    else: #Condicional si se obtiene la ubicacion
        if emergency ==1:
            SoS=int(1)
        else:
            SoS=int(0)
        EnviarDatoSigfox(data[0],data[1],data[2],data[3],
data[4],data[5],data[6],data[7],data[8],data[9],SoS)
        t=0
        p_in=Pin('P11',mode=Pin.IN,pull=Pin.PULL_UP)
        while t<660:
            t += 1
            print(t)
            if p_in.value() == 0:
                sos += 1
                pycom.rgbled(0x0F0001)#Led Naranja
                print(p_in.value())
                time.sleep(1)
            else:
                sos=0
                print(p_in.value())
                pycom.rgbled(0x010100)#Led verde
                time.sleep(1)
            if sos > 4:
                l76 = L76GNSS(py,timeout=3, buffer=512)
                coord=l76.coordinates()
                a=list(coord)
                data=Codificacion(a)
                if data == 10:
                    print('No Data')
                else:
                    emergency=1
                    SoS=int(1)
                EnviarDatoSigfox(data[0],data[1],data[2],data[3],
data[4],data[5],data[6],data[7],data[8],data[9],SoS)
```

Figura 2.14. Código de lazo while principal.

2.2.2 CONFIGURACIÓN NUBE DE SIGFOX

Para acceder a la nube de Sigfox es preciso crear una cuenta en el portal https://backend.sigfox.com detallando el propósito del proyecto y datos personales, una vez registrado es necesario vincular los dispositivos con nuestra cuenta, proceso que se realiza en el portal https://buy.sigfox.com/ donde se registra el módulo Sipy 1.0, cuando se realizó la compra del dispositivo, este incluía 1 año de conectividad Platinum en la red Sigfox por lo que se debería registrar en el portal como Kit de desarrollo, en el anexo 1 se tiene un instructivo de registro de dispositivos en el backend de Sigfox.

Una vez registrado el dispositivo y vinculado a nuestra cuenta, el dispositivo se lo puede apreciar en el portal https://backend.sigfox.com. En la figura 2.15 podemos apreciar el dispositivo registrado en el portal.

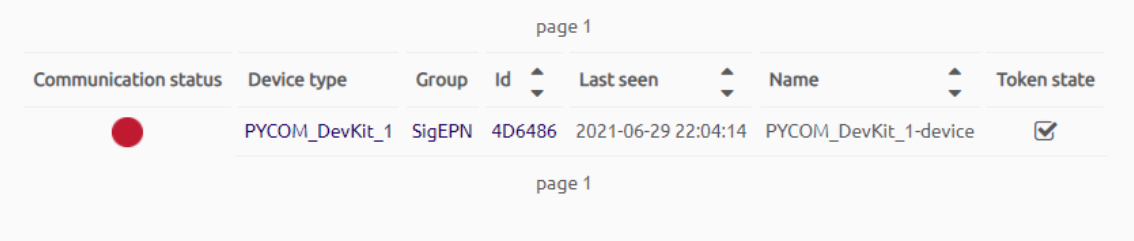


Figura 2.15. Registro de dispositivo Pycom en la plataforma Sigfox.

Una vez registrado el dispositivo podemos empezar a enviar mensajes por medio de la red Sigfox y los podemos observar en la nube de Sigfox, en la figura 2.16 podemos apreciar la presencia del payload del mensaje, el número de secuencia del mensaje, la hora y fecha del mensaje del dispositivo en la nube de Sigfox.

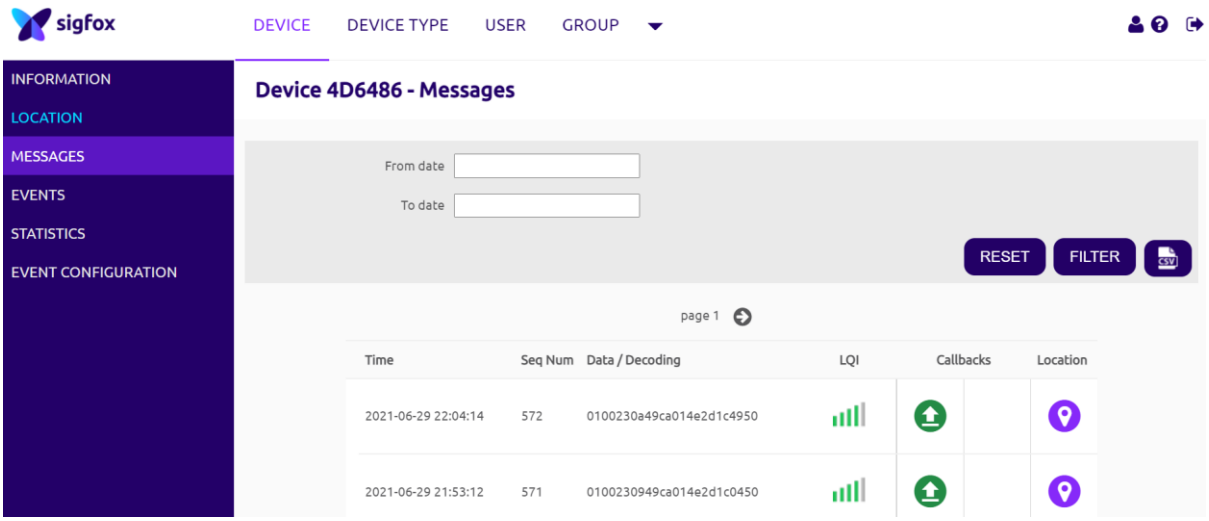


Figura 2.16. Mensajes en la Nube Sigfox.

Uno de los servicios de la nube que nos permitirá la comunicación con los servicios de AWS es el servicio de API Callback, que se debe configurar con varios datos de AWS como el ID de AWS y el rol de la API, estos datos se los obtendrá en la sección 2.2.3.1, en la figura 2.17 podemos apreciar la configuración de la API Callback en la nube de Sigfox con los valores del Id AWS, el rol asignado y la información que será enviada (device, data, time, deviceTypeId y seqNumber) en formato Json.

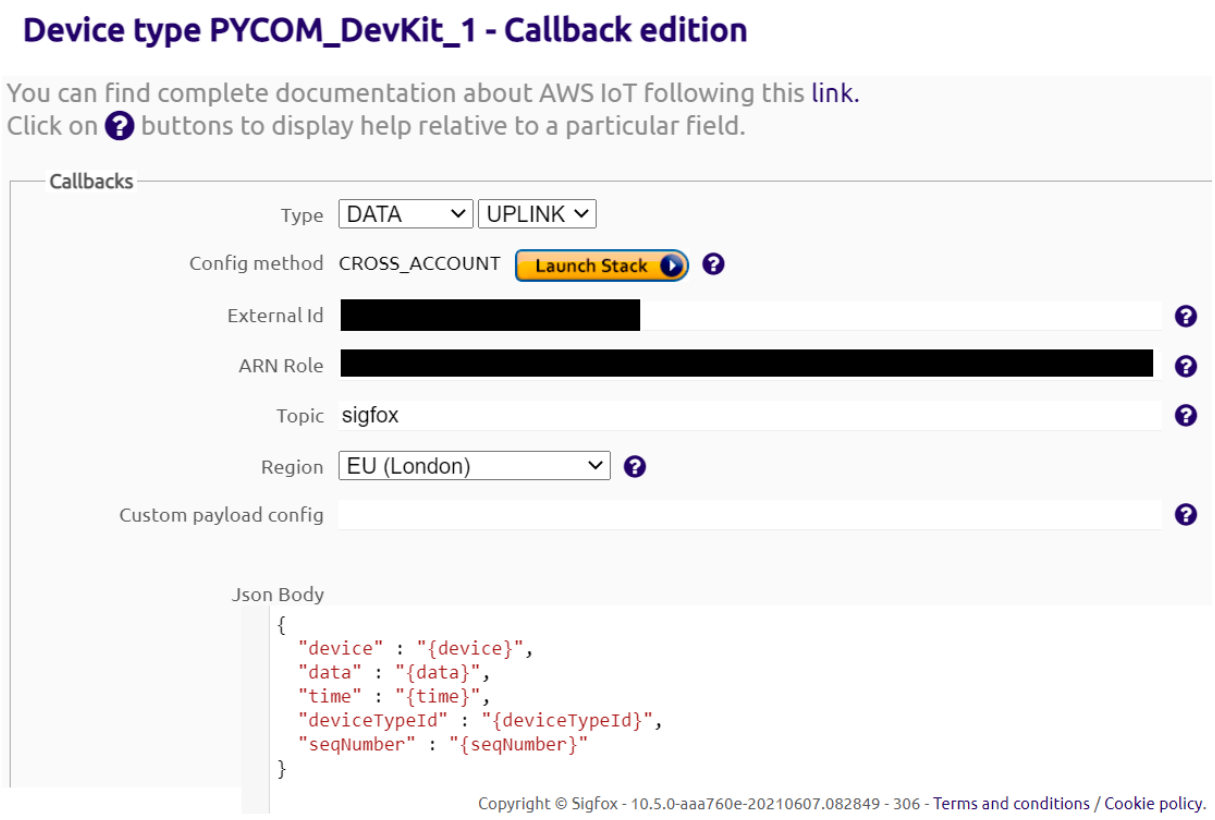


Figura 2.17. Configuración Callback en Sigfox.

2.2.3 CONFIGURACIÓN NUBE AWS

En la nube AWS se tiene que realizar varios procesos que estarán a cargo de varios servicios dentro de la nube AWS; en la figura 2.18 se aprecia los principales servicios que se requieren para el prototipo.

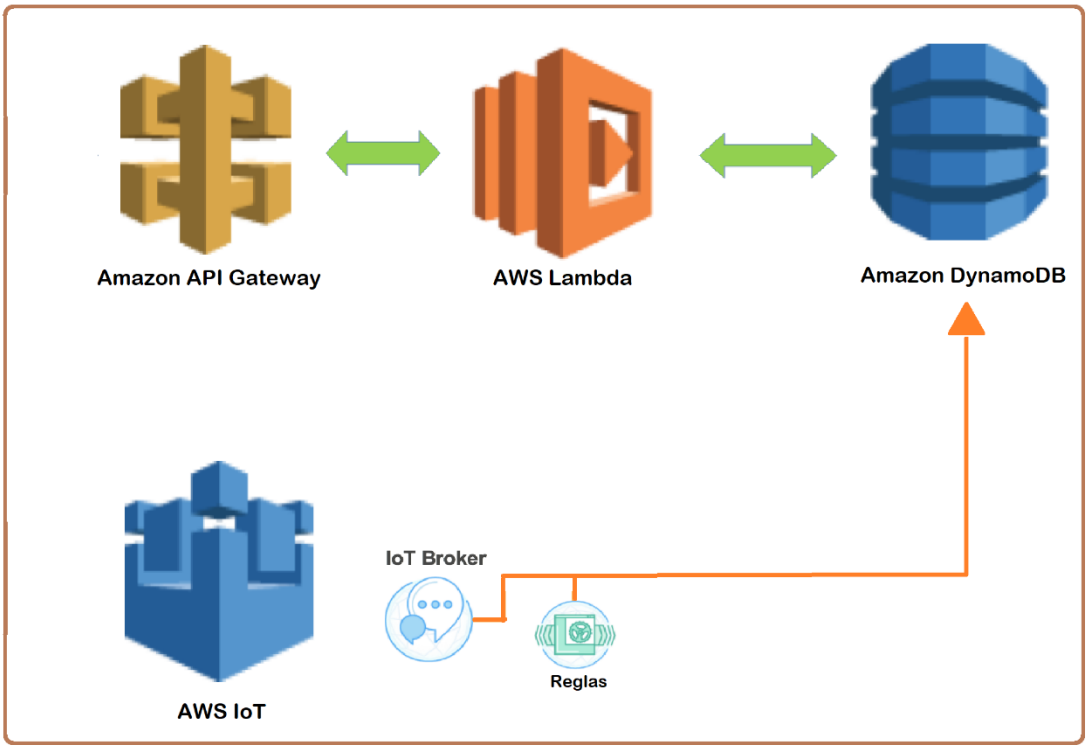


Figura 2.18. Servicios AWS.

2.2.3.1 Configuración AWS DynamoDB

En el servicio DynamoDB se utiliza para administrar los datos de la aplicación final, las tablas creadas para la aplicación serán: data_sigfox_2, devices_sigfox_2, historial_sigfox, sigfox y usuarios_sigfox.

En la figura 2.19 podemos apreciar las tablas data_sigfox_2, devices_sigfox_2, historial_sigfox, sigfox y usuarios_sigfox creadas dentro de la base de datos DynamoDB en AWS.

Nombre	Estado	Clave de partición	Clave de ordenación	Índices	Capacidad de lectura tr	Capacidad de escritura	Auto Scaling
data_sigfox_2	Activo	deviceid (Cadena)	timestamp (Cadena)	0	5	5	DISABLED
devices_sigfox_2	Activo	marca (Cadena)	deviceid (Cadena)	0	5	5	DISABLED
historial_sigfox	Activo	marca (Cadena)	timestamp (Cadena)	0	5	5	DISABLED
sigfox	Activo	deviceid (Cadena)	timestamp (Cadena)	0	5	5	DISABLED
usuarios_sigfox	Activo	Cedula (Cadena)	-	0	5	5	DISABLED

Figura 2.19. Tablas DynamoDB.

2.2.3.1.1 tabla data_sigfox_2

La tabla data_sigfox_2 se encuentra conformado por los ítems deviceid, timestamp, SOS, latitude, longitude y usuario. Esta tabla tiene el propósito de tener todos los datos decodificados de los mensajes Sigfox de todos los dispositivos que se encuentren registrados en la tabla devices_sigfox_2; los datos requeridos para la aplicación serían necesarios los datos del tiempo, el estado del botón de emergencia, la posición con latitud

y longitud y el usuario que se encuentre utilizando el dispositivo ese momento, si no se encuentra utilizando ningún usuario se coloca el valor de “0”.

deviceid	timestamp	SOS	latitude	longitude	usuario
4D6486	1625416537	80	-0.35107099999999997	-78.452812	1718349077
4D6486	1625417199	80	-0.350966	-78.452857	1718349077
4D6486	1625417863	80	-0.351043	-78.452812	1718349077
4D6486	1625418525	80	-0.35101499999999997	-78.452873	1718349077
4D6486	1625419188	80	-0.351056	-78.452865	1718349077
4D6486	1625419852	80	-0.35108599999999995	-78.452827	1234567890
4D6486	1625420215	80	-0.35089	-78.452911	1234567890
4D6486	1625421541	80	-0.329536	-78.451545	1234567890
4D6486	1625422867	80	-0.32946800000000004	-78.45156	1234567890
4D6486	1625424193	80	-0.31893	-78.450309	1234567890
4D6486	1625424856	80	-0.318921	-78.450218	1234567890
4D6486	1625425519	80	-0.318931	-78.450241	1234567890
4D6486	1625426182	80	-0.318931	-78.450241	1234567890

Figura 2.20 Elementos de la tabla data_sigfox_2.

2.2.3.1.2 tabla devices_sigfox_2

La tabla devices_sigfox_2 se encuentra conformado por los ítems marca, deviceid, SOS, latitude, longitude, modelo, status y timestamp. Esta tabla tiene el propósito de mantener actualizada la última ubicación (latitud y longitud), el estado del botón de emergencia y el status que refleja el usuario que se encuentra utilizando los dispositivos, para poder así mostrar las bicicletas disponibles en un mapa.

marca	deviceid	SOS	latitude	longitude	modelo	status	timestamp
Pycom	4D1234	80	-0.186198	-78.487304	Sipy	0	1624312262
Pycom	4D2950	80	-0.179411	-78.483516	Sipy	0	1624312262
Pycom	4D3125	80	-0.183308	-78.482327	Sipy	0	1624312262
Pycom	4D6486	80	-1.038439	-78.589607	Sipy	1234567890	1625585846
Pycom	4D6950	80	-0.128293	-78.489555	Sipy	0	1624312262
Pycom	4D7226	80	-0.351007	-78.452907	Sipy	0	1624312262
Pycom	4D7422	80	-0.351	-78.72888	Sipy	0	1624326789
Pycom	4D9192	80	-0.147294	-78.488749	Sipy	0	1624312262
Pycom	4D9472	80	-0.177058	-78.477775	Sipy	0	1624312262

Figura 2.21. Elementos de la tabla devices_sigfox_2.

2.2.3.1.3 tabla historial_sigfox

La tabla historial_sigfox se encuentra conformado por los ítems marca, timestamp, deviceid, latitud, longitud y usuario. Esta tabla tiene el propósito de almacenar la

información histórica de todos los usuarios para posteriormente poder mostrarla en la aplicación.

marca	timestamp	deviceid	latitud	longitud	usuario
Pycom	1625415210	4D6486	-0.350973	-78.452857	1718349077
Pycom	1625415874	4D6486	-0.350956	-78.452865	1718349077
Pycom	1625416537	4D6486	-0.35107099999999997	-78.452812	1718349077
Pycom	1625417199	4D6486	-0.350966	-78.452857	1718349077
Pycom	1625417863	4D6486	-0.351043	-78.452812	1718349077
Pycom	1625418525	4D6486	-0.35101499999999997	-78.452873	1718349077
Pycom	1625419188	4D6486	-0.351056	-78.452865	1718349077
Pycom	1625419852	4D6486	-0.35108599999999995	-78.452827	1234567890
Pycom	1625420215	4D6486	-0.35089	-78.452911	1234567890
Pycom	1625421541	4D6486	-0.329536	-78.451545	1234567890
Pycom	1625422867	4D6486	-0.32946800000000004	-78.45156	1234567890
Pycom	1625424193	4D6486	-0.31893	-78.450309	1234567890
Pycom	1625424856	4D6486	-0.318921	-78.450218	1234567890

Figura 2.22. Elementos de la tabla historial_sigfox.

2.2.3.1.4 tabla sigfox

La tabla sigfox se encuentra conformado por los ítems deviceid, timestamp y payload. Esta tabla cumple con la función de almacenar todos los mensajes provenientes de la nube Sigfox, en el ítem payload se encuentra codificado toda la información correspondiente a la aplicación que deberá ser decodificada.

deviceid	timestamp	payload
4D6486	1618606872628	{ "data" : { "S" : "0100210b1fca014e2b0e48fe" }, "device" : { "S" : "4D6486" }, "deviceTypeId" : { "S" : "5f245690e833d90362aa8d87" ...
4D6486	1618607072723	{ "data" : { "S" : "0100225f2dca014e2d1c60fe" }, "device" : { "S" : "4D6486" }, "deviceTypeId" : { "S" : "5f245690e833d90362aa8d87" }...
4D6486	1618607071892	{ "data" : { "S" : "0100230444ca014e2d1907fe" }, "device" : { "S" : "4D6486" }, "deviceTypeId" : { "S" : "5f245690e833d90362aa8d87" ...
4D6486	1618607075713	{ "data" : { "S" : "010023075bca014e2d185bfe" }, "device" : { "S" : "4D6486" }, "deviceTypeId" : { "S" : "5f245690e833d90362aa8d87" ...
4D6486	1618607071676	{ "data" : { "S" : "0100230814ca014e2d1710fe" }, "device" : { "S" : "4D6486" }, "deviceTypeId" : { "S" : "5f245690e833d90362aa8d87" ...
4D6486	1618607072243	{ "data" : { "S" : "010023081cca014e2d212efe" }, "device" : { "S" : "4D6486" }, "deviceTypeId" : { "S" : "5f245690e833d90362aa8d87" ...
4D6486	1618607072737	{ "data" : { "S" : "0100230821ca014e2d1b51fe" }, "device" : { "S" : "4D6486" }, "deviceTypeId" : { "S" : "5f245690e833d90362aa8d87" ...
4D6486	1618607072758	{ "data" : { "S" : "0100230837ca014e2d1c2afe" }, "device" : { "S" : "4D6486" }, "deviceTypeId" : { "S" : "5f245690e833d90362aa8d87" ...
4D6486	1618607075681	{ "data" : { "S" : "010023083cca014e2d1b14fe" }, "device" : { "S" : "4D6486" }, "deviceTypeId" : { "S" : "5f245690e833d90362aa8d87" ...
4D6486	1618607072755	{ "data" : { "S" : "010023084bca014e2d1b4afe" }, "device" : { "S" : "4D6486" }, "deviceTypeId" : { "S" : "5f245690e833d90362aa8d87" ...
4D6486	1618542133177	{ "data" : { "S" : "010023085bca014e2d1c60fe" }, "device" : { "S" : "4D6486" }, "deviceTypeId" : { "S" : "5f245690e833d90362aa8d87" ...
4D6486	1618607071888	{ "data" : { "S" : "010023085dca014e2d1c1bfe" }, "device" : { "S" : "4D6486" }, "deviceTypeId" : { "S" : "5f245690e833d90362aa8d87" ...
4D6486	1618607075416	{ "data" : { "S" : "0100230910ca014e2d1b60fe" }, "device" : { "S" : "4D6486" }, "deviceTypeId" : { "S" : "5f245690e833d90362aa8d87" ...

Figura 2.23. Elementos de la tabla Sigfox.

2.2.3.1.5 tabla usuarios_sigfox

La tabla usuarios_sigfox se encuentra conformado por los ítems Cedula, Celular, Contacto, Correo, Emergencia, Id, Nombres, Password y Status. Esta tabla tiene el propósito de

almacenar toda la información del usuario del registro inicial y de su ingreso al sistema; además se puede conocer que usuarios se encuentran haciendo uso de la bicicleta.

Cedula	Celular	Contacto	Correo	Emergencia	Id	Nombres	Password	Status
1234567890	1234567890	1234567890	ricardo.urbina@hotmail.com	OFF	0	Roberto Urbina	1234	ON
1718349077	0978600655	0978600655	rjuo240596@hotmail.com	OFF	0	Ricardo Urbina	qwert12345	ON
1720349077	0978600655	0978600655	rjuo240596@hotmail.com	OFF	0	Eduado Revelo	qwert12345	OFF
1722585443	1231232131	21	eduardo.revelo@epn.edu.ec	OFF	0	eduUpdate	123	OFF
1722585589	0983897591	0984895485	ricardo.urbina@hotmail.com	OFF	0	Pedro Urbina	12345	ON
1722585649	0983897591	0984895485	ricardo.urbina@hotmail.com	OFF	0	Ricardo Urbina	1234	ON

Figura 2.24 Elementos de la tabla usuarios_sigfox.

2.2.3.2 AWS Cloud Formation

En el servicio Cloud Formation se utiliza para por medio de la plantilla creada poder crear un rol para la recepción de los mensajes de API Callback de la nube de Sigfox y enviarlos al servicio AWS IoT, en la figura 2.25 apreciamos la creación del rol con los permisos necesario para poder leer y escribir datos, el valor de Stack ID es el valor que se debe colocar en la sección de ARN Role en la configuración de Callback en la nube Sigfox.

La creación del rol es posible verificar en el servicio IAM de AWS.

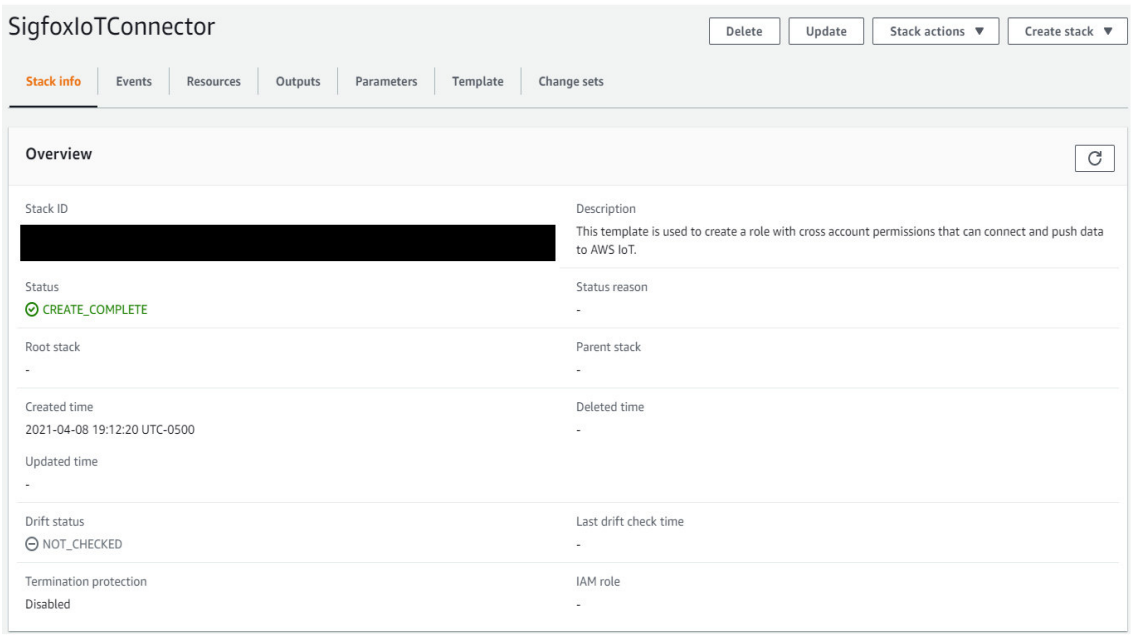


Figura 2.25 Creación de Rol en Cloud Formation.

2.2.3.3 Configuración AWS IoT Core

En el servicio de AWS IoT Core se puede monitorear el ingreso de todos los mensajes por medio de Callback desde la nube Sigfox; con este servicio se crea una regla. La regla se utiliza para guardar los mensajes en la tabla “sigfox” de DynamoDB y filtrar los mensajes,

decodificarlos y guardar los valores en la tabla “data_sigfox_2” de DynamoDB. En la figura 2.26 se puede apreciar el monitoreo de los mensajes de los meses abril, mayo y junio de 2021 en AWS IoT Core.

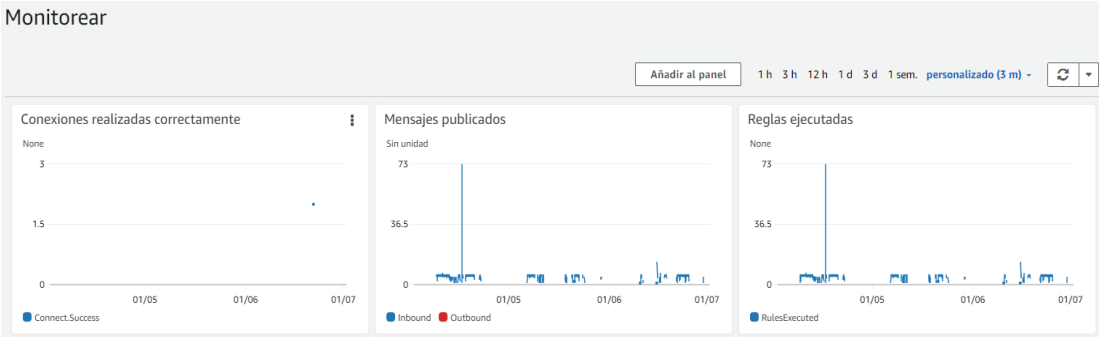


Figura 2.26 Monitoreo de los mensajes en AWS IoT Core.

2.2.3.3.1 regla AWS IoT core

La primera parte de la regla permite que todos los mensajes que lleguen se almacenen en una base de datos del servicio DynamoDB, para este propósito es necesario crear la tabla en el servicio DynamoDB que tendrá como nombre “sigfox”

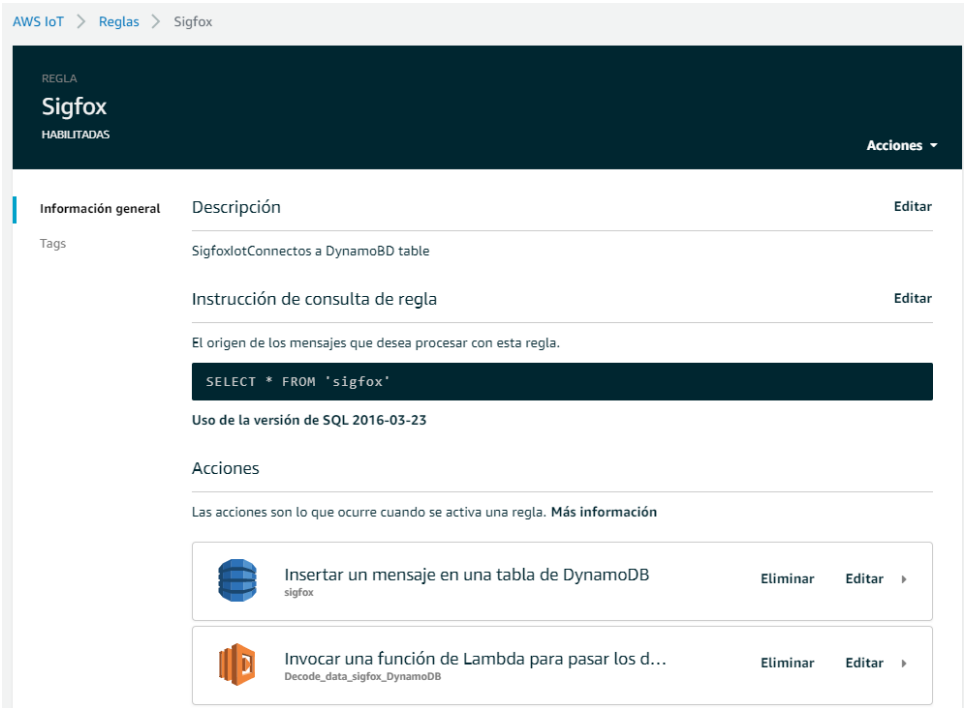


Figura 2.27. Regla en AWS IoT Core.

La segunda parte de la regla hace uso del servicio Lambda para filtrar, decodificar y guardar las variables de deviceid, timestamp, SOS, latitude y longitude en la tabla “data_sigfox_2” de la base de datos DynamoDB

En la figura 2.27 se puede apreciar la creación de la regla de nombre “Sigfox” con las 2 funcionalidades.

La función de Lambda utilizada en la regla tiene como nombre “Decode_data_sigfox_DynamoDB” que cumple con el diagrama de flujo de la figura 2.1.3.3.1.1.

La función “Decode_data_sigfox_DynamoDB” permite:

- Extraer el último mensaje guardado en la tabla Sigfox de acuerdo con el id del dispositivo.

Para extraer el último mensaje guardado se utilizó el código mostrado en la figura 2.28 en el que se realiza una petición de búsqueda en la tabla y se extrae los últimos mensajes de la tabla de acuerdo con el timestamp y el deviceid, luego se procede a escoger el último mensaje para completar el proceso.

```
    transactiondevice='4D7226'
    response = table.query(
  KeyConditionExpression=Key('deviceid').eq(str(transactiondevice))
& Key('timestamp').gt(str(ts))
    )
    response=json.dumps(response)
    objeto=json.loads(str(response))
    maxcount=objeto['Count']
    maxcount=int(maxcount)-1
    data=objeto['Items'][maxcount]['payload']['data']#el numero
cambia el numero de veces que se tiene de datos
    ts1=objeto['Items'][maxcount]['payload']['time']
    ts1=Decimal(str(ts1))
```

Figura 2.28 Código para extraer para extraer el último mensaje Sigfox.

- Filtrar los mensajes Sigfox que son exclusivos de la aplicación por medio del byte de autenticación que debe tener el valor de “CA”.

Para filtrar los mensajes Sigfox correspondientes se utiliza un condicional if y la información extraída del byte 11 del campo data del mensaje Sigfox.

- Decodificar la información del mensaje Sigfox y guardar los valores de latitud, longitud, SOS, deviceid y timestamp.

Para decodificar el mensaje Sigfox se utilizó el código mostrado en la figura 2.29 en el que se realiza varios procesos de conversión de unidades y operaciones matemáticas.

```
if byte11 == 'ca':
        byte1=data[0],data[1]
        byte1=convertTuple(byte1)
        byte1=int(byte1,16)
        byte2=data[2],data[3]
        byte2=convertTuple(byte2)
        byte2=int(byte2,16)
        byte3=data[4],data[5]
        byte3=convertTuple(byte3)
        byte3=int(byte3,16)
        byte4=data[6],data[7]
        byte4=convertTuple(byte4)
        byte4=int(byte4,16)
        byte5=data[8],data[9]
        byte5=convertTuple(byte5)
        byte5=int(byte5,16)
       longitud = longitud*(-1)
       longitud1=Decimal(str(longitud))
       byte6=data[12],data[13]
        byte6=convertTuple(byte6)
        byte6=int(byte6,16)
        byte7=data[14],data[15]
        byte7=convertTuple(byte7)
        byte7=int(byte7,16)
        byte8=data[16],data[17]
        byte8=convertTuple(byte8)
        byte8=int(byte8,16)
        byte9=data[18],data[19]
        byte9=convertTuple(byte9)
        byte9=int(byte9,16)
        byte10=data[20],data[21]
        byte10=convertTuple(byte10)
        byte10=int(byte10,16)

        bytesos=data[22],data[23]
        bytesos=convertTuple(bytesos)
        bytesos=int(bytesos,16)
        bytesos1=Decimal(str(bytesos))

        latitud=byte2+(byte3/100)+(byte4/10000)+(byte5/1000000)
        longitud=byte7+(byte8/100)+(byte9/10000)+(byte10/1000000)
        longitud1=Decimal(str(longitud))
        if byte1 == 1:
            latitud = latitud*(-1)
            latitud1=Decimal(str(latitud))
        if byte6 == 1:
            longitud = longitud*(-1)
            longitud1=Decimal(str(longitud))
```

Figura 2.29 Código para decodificar mensaje Sigfox.

- Guardar o actualizar la información decodificada en las tablas correspondientes de acuerdo con el estatus en el que se encuentre la bicicleta previamente.

Para guardar o actualizar la información de las tablas de datos para el aplicativo se realiza una consulta a la tabla “devices_sigfox_2” para conocer el estatus de nodo Sigfox mediante el deviceid.

Si el estatus tiene un valor de “0” me indica que el dispositivo se encuentra libre y no guarda el registro en el historial del cliente.

Se guarda un nuevo registro en la tabla “data_sigfox_2” con los valores actualizados de timestamp, SOS, latitud, longitud y estatus.

Se actualiza los valores de timestamp, SOS, latitud y longitud correspondiente a la tabla “devices_sigfox_2”

Todas estas funcionalidades se muestran en el código de la figura 2.30.

```
        response3=table2.query(
                KeyConditionExpression=Key('marca').eq('Pycom') &
Key('deviceid').eq(str(deviceinfos))
            )
        status=response3['Items'][0]['status']
        status1=int(status)
        if status1 != 0:
            response4 = table3.put_item(
                Item={
                    'marca':'Pycom',
                    'usuario':str(status),
                    'timestamp':str(ts1),
                    'deviceid': deviceinfos,
                    'latitud':str(latitud),
                    'longitud':str(longitud)
                    }
                )
        else:
            response4='No data'
        response1 = table1.put_item(
            Item={
                'deviceid': deviceinfos,
                'timestamp': str(ts1),
                'SOS': bytesos1,
                'latitude': latitud1 ,
                'longitude': longitud1,
                'usuario': str(status)
                }
            )
```

```
        response3=table2.query(
                KeyConditionExpression=Key('marca').eq('Pycom') &
Key('deviceid').eq(str(deviceinfos))
            )
        try:
            status=response3['Items'][0]['status']
            response2 = table2.put_item(
                Item={
                    'marca': 'Pycom',
                    'deviceid': deviceinfos,
                    'modelo':'Sipy',
                    'status':str(status),
                    'timestamp': str(ts1),
                    'SOS': bytesos1,
                    'latitude': latitud1 ,
                    'longitude': longitud1
                    }
                )
        except:
            status='0'
            response2 = table2.put_item(
                Item={

                    'marca': 'Pycom',
                    'deviceid': deviceinfos,
                    'modelo':'Sipy',
                    'status':str(status),
                    'timestamp': str(ts1),
                    'SOS': bytesos1,
                    'latitude': latitud1 ,
                    'longitude': longitud1
                    }
                )
    else:
        response='No data'
        response2='No data'
        response3='No data'
        maxcount=0
```

Figura 2.30. Código para guardar o actualizar información de las tablas.

Mediante estas reglas se automatiza la función de filtrar, decodificar y guardar o actualizar la información de los mensajes Sigfox en las tablas de DynamoDB.

2.2.3.4 Configuración AWS Api Gateway

En el servicio de AWS Api Gateway permite el intercambio de información entre la aplicación y las tablas de DynamoDB. Para la creación del servicio de Api Gateway es necesario emplear la herramienta Lambda para ejecutar el código computacional para escribir y actualizar las tablas de DynamoDB que se ocupan para la aplicación.

En la tabla 2.2 se tiene las funcionalidades de cada una de las APIs y en la figura 2.31 se aprecia la creación de estas.

Nombre	ID	Protocolo	Tipo de punto de enlace
TransactionApis_ModifyStatus	hdy6jswri1	REST	Regional
TransactionApis_Get_Devices_Info	5ueegwgd72	REST	Regional
TransactionApis_Get_History_Info	jgqsjs7u7c	REST	Regional
TransactionApis_Modify_Status_from_Users_Info	wz0z3ny13j	REST	Regional
TransactionApis_Modify_Status_from_Devices_Info	0t7pkxgzl7	REST	Regional
TransactionApis_ModifyUser	a2sfiffgii	REST	Regional
TransactionApis_NewUser	eaayz8d062	REST	Regional

Figura 2.31 APIs Api Gateway.

2.2.3.4.1 API TransactionApis_Get_Devices_Info

La API TransactionApis_Get_Devices_Info utiliza el protocolo Api Rest con el método Get, este api se encuentra vinculada con la función de Lambda Transaction_Proccessor_5 que permite obtener toda la información de la tabla devices_sigfox_2 y enviarla en formato json. En la figura 2.32 se puede apreciar el código utilizado en la función en el cual se realiza la consulta a la tabla devices_sigfox_2 y se construye el mensaje json para ser enviado cuando sea solicitado.

El enlace para usar esta API es: https://5ueegwgd72.execute-api.eu-west-2.amazonaws.com/Devices/transactions

```
import json
import boto3
from boto3.dynamodb.conditions import Key
from decimal import Decimal
dynamodb = boto3.resource('dynamodb')
client=boto3.client('dynamodb')
table = dynamodb.Table('devices_sigfox_2')
print('Loading function')
def lambda_handler(event, context):
        #1. Parse out query string params
        response = table.query(
                KeyConditionExpression=Key('marca').eq('Pycom')
        )
```

```
        objeto=response['Items']
        count=response['Count']
        #2. Construct the body of the response object
        transactionResponse = {}
        transactionResponse['data'] = str(objeto)
        #3. Construct http response object
        responseObject = {}
        responseObject['statusCode'] = 200
        responseObject['headers'] = {}
        responseObject['headers']['Content-Type'] =
'application/json'
        responseObject['body'] = json.dumps(transactionResponse)
        #4. Return the response object
        return responseObject
```

Figura 2.32 Código función Transaction_Proccessor_5.

2.2.3.4.2 API TransactionApis_Get_History_Info

La API TransactionApis_Get_History_Info utiliza el protocolo Api Rest con el método Get, este api se encuentra vinculada con la función de Lambda Transaction_Proccessor_9 que permite obtener la información histórica de un usuario por medio de su número de cédula y enviarla en formato json. En la figura 2.33 se puede apreciar el código utilizado en la función en el cual se realiza la consulta a la tabla historial_sigfox en la que se encuentra guardado el historial de todos los usuarios y se filtra la información por medio del número de cédula del usuario que se requiere la información del historial.

```
import json
import boto3
from boto3.dynamodb.conditions import Key
from decimal import Decimal
dynamodb = boto3.resource('dynamodb')
client=boto3.client('dynamodb')
table = dynamodb.Table('historial_sigfox')
print('Loading function')
def lambda_handler(event, context):
    usuario_in = event['queryStringParameters']['usuario_in']
    data = table.scan()
    count = data['Count']
    data1 = data['Items']
    ref=0
    while ref < count:
        element=data1[ref]
        user = element['usuario']
        if user == usuario_in:
            ref=ref+1
        else:
            data1.pop(ref)
            ref=ref
            count=count-1
```

```
    transactionResponse = {}
    transactionResponse['data'] = str(data1)
    #3. Construct http response object
    responseObject = {}
    responseObject['statusCode'] = 200
    responseObject['headers'] = {}
    responseObject['headers']['Content-Type'] = 'application/json'
    responseObject['body'] = json.dumps(transactionResponse)
    #4. Return the response object
    return responseObject
```

Figura 2.33 Código de la función Transaction_Proccessor_9.

El enlace para usar esta API es: https://jgqsjs7u7c.execute-api.eu-west-2.amazonaws.com/Get_History/transactions?usuario_in=´**Cédula**´

2.2.3.4.3 API TransactionApis_Modify_Status_from_Users_Info

La API TransactionApis_Modify_Status_from_Users_Info utiliza el protocolo Api Rest con el método Get, este api se encuentra vinculada con la función de Lambda Transaction_Proccessor_7 que permite obtener toda la información de la tabla usuarios_sigfox y enviarla en formato json. En la figura 2.34 se puede apreciar el código utilizada en esta función en el cual se realiza la consulta a la tabla usuarios_sigfox y se construye el mensaje json para ser enviado cuando sea solicitado.

```
import json
import boto3
from boto3.dynamodb.conditions import Key
from decimal import Decimal
dynamodb = boto3.resource('dynamodb')
client=boto3.client('dynamodb')
table = dynamodb.Table('usuarios_sigfox')
print('Loading function')
def lambda_handler(event, context):
        #1. Parse out query string params
        data = table.scan()
        objeto=data['Items']
        #2. Construct the body of the response object
        transactionResponse = {}
        transactionResponse['data'] = str(objeto)
        #3. Construct http response object
        responseObject = {}
        responseObject['statusCode'] = 200
        responseObject['headers'] = {}
        responseObject['headers']['Content-Type'] = 'application/json'
        responseObject['body'] = json.dumps(transactionResponse)
        #4. Return the response object
        return responseObject
```

Figura 2.34 Código de la función Transaction_Proccessor_7.

El enlace para usar esta API es: https://wz0z3ny13j.execute-api.eu-west-2.amazonaws.com/ModifyStatus/transactions

2.2.3.4.4 API TransactionApis_Modify_Status_from_Devices_Info

La API TransactionApis_Modify_Status_from_Devices_Info utiliza el protocolo Api Rest con el método Get, este api se encuentra vinculada con la función de Lambda Transaction_Proccessor_6 que permite modificar el ítem status de la bicicleta de la tabla devices_sigfox_2 al número de cédula del usuario que alquilo la bicicleta y permite cambiar el ítem status de la tabla devices_sigfox_2 a "0" si la bicicleta fue devuelta. En la figura 2.35 se puede apreciar el código utilizado en la función en el cual se realiza la consulta a la tabla devices_sigfox_2 y se filtra la información por medio de la información del deviceid proporcionado, una vez que se obtiene la información se actualiza la información del estatus con el valor proporcionado.

```
import json
import boto3
from boto3.dynamodb.conditions import Key
from decimal import Decimal
dynamodb = boto3.resource('dynamodb')
client=boto3.client('dynamodb')
table = dynamodb.Table('devices_sigfox_2')
print('Loading function')
def lambda_handler(event, context):
        Cedula = event['queryStringParameters']['Cedula']
        DeviceId = event['queryStringParameters']['DeviceId']
        response = table.query(
                KeyConditionExpression=Key('marca').eq('Pycom') &
Key('deviceid').eq(str(DeviceId))
        )
        try:
            marca=response['Items'][0]['marca']
            deviceid=response['Items'][0]['deviceid']
            SOS=response['Items'][0]['SOS']
            latitude=response['Items'][0]['latitude']
            longitude=response['Items'][0]['longitude']
            modelo=response['Items'][0]['modelo']
            timestamp=response['Items'][0]['timestamp']
            status=str(Cedula)
            mensaje='Cambiado Status'
            response2 = table.put_item(
                Item={
                    'marca': str(marca),
                    'deviceid': str(deviceid),
                    'modelo':str(modelo),
                    'status':str(status),
                    'timestamp': str(timestamp),
                    'SOS': str(SOS),
                    'latitude': str(latitude) ,
                    'longitude': str(longitude)
                }
                )
        except:
            mensaje='Error'
```

```
        #2. Construct the body of the response object
        transactionResponse = {}
        transactionResponse['Status'] = status
        transactionResponse['mensaje'] = mensaje
        #3. Construct http response object
        responseObject = {}
        responseObject['statusCode'] = 200
        responseObject['headers'] = {}
        responseObject['headers']['Content-Type']='application/json
        responseObject['body'] = json.dumps(transactionResponse)
        #4. Return the response object
        return responseObject
```

Figura 2.35. Código de la función Transaction_Proccessor_7.

El enlace para usar esta API es: https://0t7pkxgzl7.execute-api.eu-west-2.amazonaws.com/Modificar_Status/transactions?Cedula=***Cédula_0***&DeviceId=***DeviceId***

2.2.3.4.5 API TransactionApis_ModifyStatus

La API TransactionApis_ModifyUser utiliza el protocolo Api Rest con el método Get, este api se encuentra vinculada con la función de Lambda Transaction_Proccessor_4 que permite modificar los ítems id y status del usuarios de la tabla usuarios_sigfox con el número de cédula del usuario que alquilo la bicicleta, si alquila la bicicleta el id tiene el valor del deviceid de la bicicleta y el status cambia de OFF a ON, si entrega la bicicleta el id tiene el valor de "0" y el status cambia de ON a OFF. En la figura 2.36 se puede apreciar el código utilizado en la función en el cual se realiza la consulta a la tabla usuarios_sigfox y se filtra con el número de cédula la información del usuario y se actualiza la información con los nuevos datos de estatus y el id.

El enlace para usar esta API es: https://hdy6jswri1.execute-api.eu-west-2.amazonaws.com/ModifyStatus/transactions?Cedula=***Cédula***&Status=***ON_OFF***&Id=***DeviceId_0***

```
import json
import boto3
from boto3.dynamodb.conditions import Key
from decimal import Decimal
dynamodb = boto3.resource('dynamodb')
client=boto3.client('dynamodb')
table = dynamodb.Table('usuarios_sigfox')
print('Loading function')
def lambda_handler(event, context):
```

```
        #1. Parse out query string params
        Cedula = event['queryStringParameters']['Cedula']
        Status = event['queryStringParameters']['Status']
        Id = event['queryStringParameters']['Id']
        #2. Construct the body of the response object
        transactionResponse = {}
        response=table.query(
            KeyConditionExpression=Key('Cedula').eq(str(Cedula))
            )
        count=response['Count']
        if int(count)>0:
            Cedula=response['Items'][0]['Cedula']
            Celular=response['Items'][0]['Celular']
            Contacto=response['Items'][0]['Contacto']
            Correo=response['Items'][0]['Correo']
            Nombres=response['Items'][0]['Nombres']
            Password=response['Items'][0]['Password']
            Emergencia=response['Items'][0]['Emergencia']
            transactionResponse['message']='Usuario modificado
Status e Id'
            response1=table.put_item(
                Item={
                    'Cedula':str(Cedula),
                    'Celular':str(Celular),
                    'Contacto':str(Contacto),
                    'Correo':str(Correo),
                    'Emergencia':str(Emergencia),
                    'Nombres':str(Nombres),
                    'Password':str(Password),
                    'Status':str(Status),
                    'Id':str(Id)
                }
                )
            transactionResponse['Cedula']=str(Cedula)
            transactionResponse['Nombres']=str(Nombres)
            transactionResponse['Correo']=str(Correo)
            transactionResponse['Respuesta']='True'

        else:
            transactionResponse['message']='Usuario no se encuentra
registrado'
            transactionResponse['Respuesta']='False'
        #3. Construct http response object
        responseObject = {}
        responseObject['statusCode'] = 200
        responseObject['headers'] = {}
        responseObject['headers']['Content-Type'] =
'application/json'
        responseObject['body'] = json.dumps(transactionResponse)
        #4. Return the response object
        return responseObject
```

Figura 2.36 Código de la función Transaction_Proccessor_4.

2.2.3.4.6 API TransactionApis_NewUse

La API TransactionApis_NewUser utiliza el protocolo Api Rest con el método Get, este api se encuentra vinculada con la función de Lambda Transaction_Proccessor_2 cuya principal función es la creación de un nuevo usuario con sus datos personales como nombres, correo, password, celular, contacto, emergencia y status; también tiene la funcionalidad de detectar si ya se encuentra creado un usuario con el mismo número de cédula para evitar usuarios repetidos con el mismo número de cédula y también tiene la funcionalidad de con la opción de "prueba" en nombres y un número de cédula de conocer si ya se encuentra registrado el usuario con ese número de cédula. En la figura 2.37 se puede apreciar el código utilizado en la función en el cual se obtiene los datos de nombres, correo, contraseña, número celular, número de contacto, emergencia y estatus del api; una vez que se obtiene los datos se procede a crear las 3 funcionalidades de este api. La primera funcionalidad es la de crear un usuario siempre y cuando no se tenga registrado ningún usuario con ese número de cédula, en el caso de que se encuentre creado algún usuario con ese número de cédula el api devuelve el mensaje "Usuario ya se encuentra creado". En el caso de que en el api se envíe la palabra "prueba" como argumento de nombres se puede realizar la comprobación de si un usuario se encuentra creado o no en la base de datos.

El enlace para usar esta API es: https://eaayz8d062.execute-api.eu-west-2.amazonaws.com/User/transactions?Cedula=**cédula**&Nombres=**nombres-completos_prueba**&Correo=**correo**&Password=**contraseña**&Celular=**número-de-celular**&Contacto=**número-de-emergencia**&Emergencia=OFF&Status=OFF

```
import json
import boto3
from boto3.dynamodb.conditions import Key
from decimal import Decimal
dynamodb = boto3.resource('dynamodb')
client=boto3.client('dynamodb')
table = dynamodb.Table('usuarios_sigfox')

def lambda_handler(event, context):
        #1. Parse out query string params
        Cedula = event['queryStringParameters']['Cedula']
        Nombres = event['queryStringParameters']['Nombres']
        Correo = event['queryStringParameters']['Correo']
        Password = event['queryStringParameters']['Password']
        Celular = event['queryStringParameters']['Celular']
        Contacto = event['queryStringParameters']['Contacto']
        Emergencia = event['queryStringParameters']['Emergencia']
        Status = event['queryStringParameters']['Status']
```

```
        #2. Construct the body of the response object
        transactionResponse = {}
        response=table.query(
            KeyConditionExpression=Key('Cedula').eq(str(Cedula))
            )
        count=response['Count']
        probe='prueba'
        if str(Nombres)==probe and int(count)>0:
                transactionResponse['message']='Usuario ya se encuentra
creado'
                Password=response['Items'][0]['Password']
                transactionResponse['Password']=str(Password)
                transactionResponse['Respuesta']='True'
        elif int(count)>0:
            Nombres=response['Items'][0]['Nombres']
            Cedula=response['Items'][0]['Cedula']
            Correo=response['Items'][0]['Correo']
            Password=response['Items'][0]['Password']
            transactionResponse['message']='Usuario ya se encuentra
creado'
            transactionResponse['Cedula']=str(Cedula)
            transactionResponse['Nombres']=str(Nombres)
            transactionResponse['Correo']=str(Correo)
            transactionResponse['Password']=str(Password)
            transactionResponse['Respuesta']='false'

        elif int(count)==0 and str(Nombres)!=probe:
            transactionResponse['message']='Usuario creado
exitosamente'
            transactionResponse['Cedula'] = Cedula
            transactionResponse['Nombres'] = Nombres
            transactionResponse['Correo']= Correo
            transactionResponse['Respuesta']='true'
            response1=table.put_item(
                Item={
                    'Cedula':str(Cedula),
                    'Celular':str(Celular),
                    'Contacto':str(Contacto),
                    'Correo':str(Correo),
                    'Emergencia':str(Emergencia),
                    'Nombres':str(Nombres),
                    'Password':str(Password),
                    'Status':str(Status),
                    'Id':str(0)
                }
                )
        else:
                transactionResponse['message']='Usuario no se encuentra
creado'
                transactionResponse['Respuesta']='False'
```

```
    #3. Construct http response object
    responseObject = {}
    responseObject['statusCode'] = 200
    responseObject['headers'] = {}
    responseObject['headers']['Content-Type'] = 'application/json'
    responseObject['body'] = json.dumps(transactionResponse)

    #4. Return the response object
    return responseObject
```

Figura 2.37 Código de la función Transaction_Proccessor_2.

2.2.4 APLICACIÓN DE USUARIO

La aplicación de usuario se realiza en base al diagrama de máquina de estados de la figura 2.9, en el que se diseña la secuencia de eventos que sigue el prototipo.

Para el desarrollo de la aplicación del prototipo se utiliza el lenguaje de programación de JavaScript con la librería React.js. La aplicación permite la interacción con las tablas creadas en Amazon Web Service mediante Apis con la herramienta API Fetch.

2.2.4.1 Registro de usuario

Para el registro de usuarios se crea un formulario que contengan los datos necesarios del usuario, se validan los datos ingresados por el usuario para que no exista inconsistencias en las tablas de AWS. En este componente de la aplicación de usuario se hace uso de la API TransactionApis_NewUser para poder realizar el registro del nuevo usuario en la tabla usuarios_sigfox. En la figura 2.38 se aprecia un extracto del código utilizado para validar los datos y utilizar la API TransactionApis_NewUser con los datos ingresados en el formulario.

```
const RegitrarseForm = () => {

    const submitHandler = (e) => {
        if(form.password === form.password2){
            let url = `https://eaayz8d062.execute-api.eu-west-
2.amazonaws.com/User/transactions?Cedula=${form.cedula}&Nombres=${for
m.nombre}&Correo=${form.correo}&Password=${form.password}&Celular=${f
orm.celular}&Contacto=${form.contacto}&Emergencia=OFF&Status=OFF`;
            console.log(url);
            helpHttp().get(url).then((res)=>{
                if(!res.err){
                    setResponse(res);
                    console.log(res);
                    setError(null);
                    //alert(response.message);
                }else{
                    setResponse(null);
                            setError(res);
                }
            });
        }else{
            e.preventDefault();
            e.stopPropagation();
            alert("Deben coincidir las contraseñas");
        }
    }
```

Figura 2.38 Código del aplicativo para registro de nuevo usuario.

2.2.4.2 Inicio de sesión

Para el inicio de sesión se hace uso de la API TransactionApis_NewUser con la opción de solo retornar información del usuario con el número de cédula y en nombre con la opción "prueba" para validar la información ingresada en el formulario y proceder a dar acceso al usuario. En la figura 2.39 se aprecia el código utilizado para validar el ingreso de los usuarios al portal.

```
const LoginForm = () => {
    let history = useHistory();
    const [form, setForm] = useState(initialForm);
    const [response, setResponse] = useState(null);
    const [isValidate, setIsValidate] = useState(false);
    const [path, setPath] = useState(null);
    const [error, setError] = useState(null);
    const singInHandler = ()=>{
        let url =`https://eaayz8d062.execute-api.eu-west-
2.amazonaws.com/User/transactions?Cedula=${form.cedula}&Nombres=pru
eba&Correo=&Password=&Celular=&Contacto=&Emergencia=&Status=`;
        console.log(url);
                helpHttp().get(url).then((res)=>{
            if(!res.err){
                setResponse(res);
                console.log(res);
                console.log(res.Password)
                if(res.Password == form.password){
                    setIsValidate(true);
                    setPath(`/home/${form.cedula}`);
                }
                setError(null);
            }else{
                setResponse(null);
                setError(res);
            }
        })
    }
```

Figura 2.39 Código del aplicativo para Inicio de sesión en el portal con validación.

2.2.4.3 Reservar bicicleta

Una vez se tenga acceso al portal se podrá escoger la opción de Reservar bicicleta, en el que se observa en un mapa las distintas bicicletas dentro del prototipo, para esto se accede a la información de la tabla devices_sigfox_2 con la API TransactionApis_Get_Devices_Info. Las bicicletas se muestran dentro del mapa como marcadores. En la figura 2.40 se aprecia como se obtiene la información de la API TransactionApis_Get_Devices_Info.

```
const Map = ({cedula}) => {
    const [status, setStatus] = useState(null);
    let url2="https://5ueegwgd72.execute-api.eu-west-
2.amazonaws.com/Devices/transactions";
    const [data, setData] = useState(null);
    const [error, setError] = useState(null);
    useEffect(() => {
        helpHttp().get(url2).then((res) => {
            if(!res.err){

setData(JSON.parse(res.data.replace(/Decimal[(]/g,"").replace(/[)]/g
,"").replace(/['']/g,`"`)));
                setError(null);
            }else{
                setData(null);
                            setError(res);
            }
        });
    }, [status]);
    return (
        <div>
            { data &&
                <ApiMap
                    data={data}
                    cedula={cedula}
                    status={status}
                    setStatus={setStatus}
                />
            }
        </div>
    )
}
export default Map;
```

Figura 2.40 Código del aplicativo para obtener datos de la tabla devices_sigfox_2.

Una vez obtenida la información de las últimas ubicaciones de los dispositivos se procede a graficar marcadores con sus coordenadas en el mapa de Google mediante la API de Google Maps.

Para realizar la reserva se tiene un botón de reservar cuando se haya seleccionado una de las bicicletas disponibles, para este propósito se hace uso de la API TransactionApis_Modify_Status_from_Users_Info y de la API TransactionApis_ModifyUser. En la figura 2.41 se aprecia un extracto del código utilizado para modificar el estatus del usuario y del dispositivo cuando se realiza una reserva.

```
const reservarHandler = (e) => {
        console.log(validacion);
        if(validacion === true){
            alert("Ya tiene una reserva Activa");
        }else{
            //url para cambiar el status de la bicicleta
            var aleatorio = Math.random();
            let url = `https://0t7pkxgzl7.execute-api.eu-west-
2.amazonaws.com/Modificar_Status/transactions?Cedula=${cedula}&Dev
iceId=${bicycle.deviceid}`;
            //URL para cambiar el estado del usuario
            let url2 = `https://hdy6jswri1.execute-api.eu-west-
2.amazonaws.com/ModifyStatus/Transaction_Proccessor_4?Cedula=${ced
ula}&Status=ON&Id=${bicycle.deviceid}`;
            helpHttp().get(url).then((res)=>{
                if(!res.err){
                    setStatus(aleatorio);
                    setResponse(res);
                    setError(null);
                    helpHttp().get(url2).then((res)=>{
                        if(!res.err){
                            setResponse2(res);
                            setError(null);
                     alert("Reserva exitosa");
                        }else{
                            setResponse2(null);
                            setError(res);
                        }
                    })
                }else{
                    setResponse(null);
                    setError(res);
                }
            });
        }
```

Figura 2.41 Código del aplicativo para reservar una bicicleta.

2.2.4.3.1 configuración Api de Google maps

Para obtener el acceso al api de Google maps es necesario crear un usuario en la nube de Google y solicitar el acceso al api de Google maps, una vez que se adquiere la clave del api se la puede verificar en el portal de la nube de Google, tal y como se aprecia en la figura 2.42.

Claves de API

Nombre	Fecha de creación	Restricciones	Clave	Acciones
Clave de API 1	12 may. 2021	Ninguna	AIzaSyDbeD...yVg40z-6v4	

Figura 2.42. Clave de Api de Google maps.

Una vez obtenido la clave de api se procede a crear el elemento mapa dentro del aplicativo para que se pueda mostrar las ubicaciones de las bicicletas en el mapa, en este elemento se puede especificar el tamaño del mapa, su resolución y los marcadores que se presentarán en el mapa.

2.2.4.4 Quitar reserva

Para quitar la reserva de una bicicleta se hace uso de las mismas Apis para la reserva, solo que se debe colocar el valor de "0" en las variables correspondientes. En la figura 2.43 se aprecia el código utilizado para quitar la reserva de un usuario.

```
    const quitarReservaHandler = (e) => {
        var aleatorio = Math.random();
        let url = `https://0t7pkxgzl7.execute-api.eu-west-
2.amazonaws.com/Modificar_Status/transactions?Cedula=${cedula}&DeviceI
d=${bicycle.deviceid}`;
        let url2 = `https://hdy6jswri1.execute-api.eu-west-
2.amazonaws.com/ModifyStatus/Transaction_Proccessor_4?Cedula=${cedula}
&Status=OFF&Id=0`;
        helpHttp().get(url).then((res)=>{
            if(!res.err){
                setStatus(aleatorio);
                setResponse(res);
                setError(null);
                helpHttp().get(url2).then((res) => {
                    if(!res.err){
                        setResponse2(res);
                        setError(null);
                        alert("Reserva finalizada");
                    }else{
                        setResponse2(null);
                        setError(res);
                        alert("Error");
                    }
                })
            }else{
                setResponse(null);
                setError(res);
            }
        });
    }
```

Figura 2.43 Código del aplicativo para quitar la reserva una bicicleta.

2.2.4.5 Historial

El aplicativo permite visualizar en una tabla y un mapa el historial de cada usuario, para este propósito se hace uso de la API TransactionApis_Get_History_Info y se muestra la información correspondiente a cada usuario. En la figura 2.44 se aprecia el código utilizado para mostrar la información correspondiente al historial de cada usuario.

```
    const Historial = () => {
     const [data, setData] = useState([]);
     const [response, setResponse] = useState(null);
     const [error, setError] = useState(null);
     let history = useHistory();
     let {cedula}= useParams();
     const volverHandler = () => {
         history.push(`/home/${cedula}`)
     }
       useEffect(()=>{
               console.log(cedula);
               let url = "https://jgqsjs7u7c.execute-api.eu-west-
       2.amazonaws.com/Get_History/transactions?usuario_in=" +
       cedula;
               //console.log(url);
               helpHttp().get(url).then((res) => {
                   if(!res.err){

       setResponse(JSON.parse(res.data.replace(/Decimal[(]/g,"").repl
       ace(/[)]/g,"").replace(/[']/g,`"`)));
                       console.log(response)
                       setError(null);
                   }else{
                       setResponse(null);
                       setError(res);
                   }
               });
           },[]);

     return (
         <div>
             <Container>
                 <Row>
                     <Col lg={10} md={10} sm={10} xs={10}>
                         <h1>HISTORIAL</h1>
                     </Col>
                 </Row>
                 <Row>
                     <Col lg={10} md={10} sm={10} xs={10}>
                         {!response ? (<Alert variant="primary" >No
tiene datos </Alert>) : (<h1>{<TablaHistorial response={response}
/>}</h1>) }
                     </Col>
                 </Row>
                 <Row>
```

```
                    <Col lg={10} md={10} sm={10} xs={10}>
                        <h1>{!response ? (<Alert variant="primary"
>No tiene datos</Alert>) : (<h1>{<MapaH response={response}
/>}</h1>) }</h1>
                    </Col>
                </Row>
                <Row>
                        <Col lg={2} md={2} sm={2} xs={2}>
                            <Button variant="success" size="lg"
onClick={volverHandler}> Volver </Button>
                        </Col>
                </Row>
            </Container>
        </div>
    )
}
export default Historial;
```

Figura 2.44 Código del aplicativo para mostrar el historial de un usuario.

2.3 FASE DE PRUEBAS

Las pruebas del prototipo de rastreo de bicicletas con tecnología Sigfox se realizaron en 3 lugares distintos de la provincia de Pichincha, para lo cual se hizo un cambio en la configuración del nodo para que envíe los mensajes cada minuto y no cada 6 minutos como se tenía previsto debido al número limitado de mensajes diarios que se puede enviar. El principal propósito para cambiar el intervalo de tiempo es para poder comprobar la recepción de los mensajes.

El aplicativo se ejecutará en la laptop y se verificará el funcionamiento de todos sus componentes, todas las pruebas se realizarán con el dispositivo Sipy 1.0 con ID 4D7226 que ya se encuentra configurado en la nube Sigfox y colocado en la bicicleta como se aprecia en la figura 2.45.

Figura 2.45 Nodo Sigfox colocado en la bicicleta.

El primer lugar donde se realizó las pruebas del prototipo fue en el parque de la Carolina ubicado en el Distrito Metropolitano de Quito.

El segundo lugar donde se realizó las pruebas del prototipo fue en Tumbaco.

El tercer lugar donde se realizó las pruebas del prototipo fue en Sangolquí.

Para cada uno de los diferentes lugares que se realizará las pruebas se crearán usuarios nuevos para cada sitio.

2.3.1 PRUEBA 1 PARQUE DE LA CAROLINA

La prueba dentro del parque del parque de la Carolina se realizará un recorrido; el mapa de cobertura Sigfox dentro del parque se aprecia en la figura 2.46 y nos indica que tenemos cobertura de 3 estaciones base en el parque.

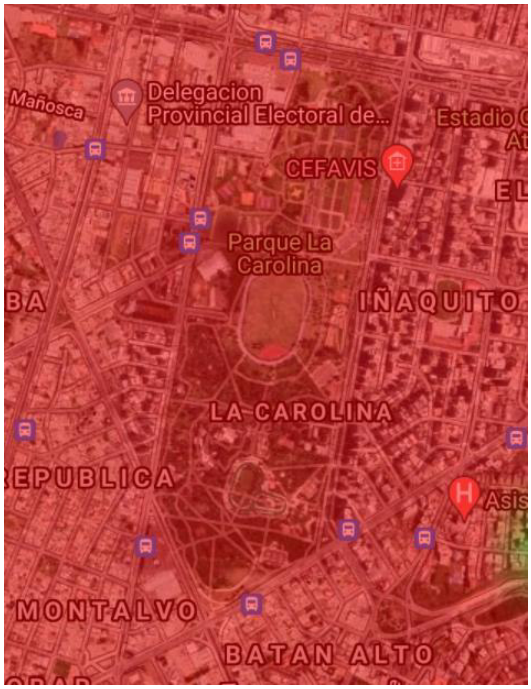


Figura 2.46. Cobertura Sigfox en el parque de la Carolina.

2.3.1.1 Ruta dentro del Parque la Carolina

La ruta dentro del parque de la Corolina se encuentra descrito en la figura 2.47 que sale del estacionamiento jardín botánico de Quito, recorre la ruta del parque la Carolina y termina en el estacionamiento del jardín botánico, el recorrido tiene una longitud aproximada de 4 kilómetros.

Figura 2.47. Ruta Parque la Carolina.

2.3.1.2 Creación de un nuevo usuario

En primer lugar, se registra un usuario nuevo dentro del aplicativo como se aprecia en la figura 2.48

Figura 2.48. Registro de usuario.

Se comprueba la creación del usuario en la tabla usuarios_sigfox y se lo puede apreciar en la figura 2.49.

Cedula	Celular	Contacto	Correo	Emergen...	Id	Nombres	Password	Status
0203040807	0995678912	0995678912	CAROLINA@GMAIL.COM	OFF	0	PARQUE CAROLINA 2	prueba1234	OFF

Figura 2.49. Registro de un nuevo usuario en la tabla usuarios_sigfox.

2.3.1.3 Inicio de Sesión

En segundo lugar, iniciamos sesión con el número de cédula y la contraseña correspondiente al usuario "PARQUE CAROLINA 2" que se lo puede apreciar en la figura 2.50.

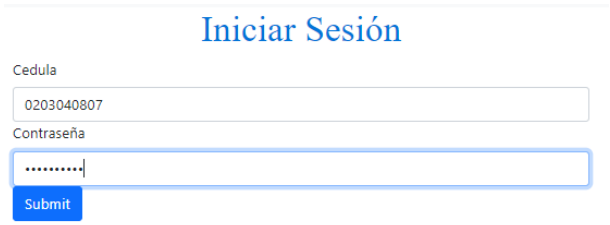
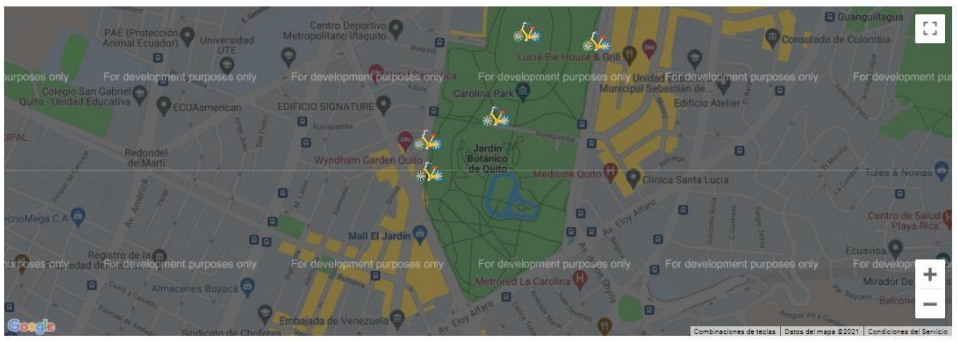


Figura 2.50. Inicio de Sesión del usuario "PARQUE CAROLINA 2"

2.3.1.3.1 reservar

En el caso de una autenticación exitosa, procedemos a entrar en la opción Reservar y seleccionamos la bicicleta que se encuentra ubicada en el jardín botánica dentro del parque de la Carolina y comprobamos que sea el ID 4D7226 y aplastamos el botón reservar. En la figura 2.51 se puede apreciar el mapa de las ubicaciones de las diferentes bicicletas y la selección de la bicicleta que se encuentra en el jardín botánico con el ID 4D7226.



Id dispositivo	Latitud	Longitud	SOS	Fecha	
4D7226	-0.1855	-78.485275	80	9/12/2021 12:25:11	Reservar

Volver

Figura 2.51. Reserva de bicicleta por el usuario "PARQUE CAROLINA 2".

El aplicativo mostrará una notificación de reserva exitosa como se aprecia en la figura 2.52.

Figura 2.52. Notificación de reserva exitosa.

2.3.1.3.2 quitar reserva

En el caso de tener una reserva activa, se puede pulsar en el botón de "Quitar reserva" y se procederá a quitar la reserva que se encuentre activa para ese usuario, en el caso de la prueba se activó el botón de "Quitar reserva" una vez finalizado el recorrido. En la figura 2.53 se aprecia el mensaje de notificación mostrado por la aplicación.

Figura 2.53 Notificación de reserva finalizada.

2.3.1.3.3 historial.

En la opción de "Historial" se muestra la información de la ubicación del usuario en el mapa y en la tabla se muestra las ubicaciones con la hora en la que se obtuvo la ubicación, en la figura 2.54.a se tiene la información del historial del usuario cunado se tenía tan solo 5 mensajes y en la figura 2.54.b se tiene la información del historial cuando se tenía 10 mensajes.

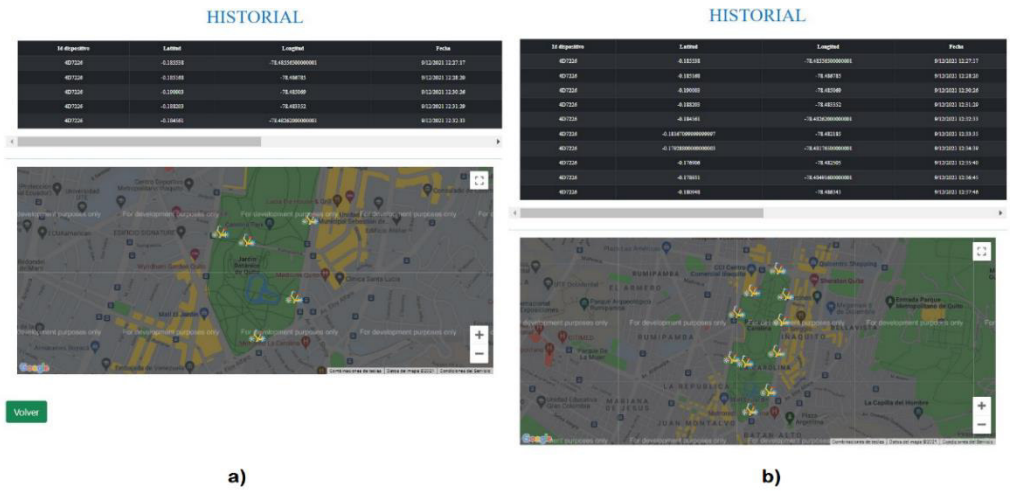


Figura 2.54. Historial parcial del usuario "PARQUE CAROLINA 2"

En la figura 2.55 se aprecia el historial final del recorrido culminado en el parque de la Carolina; El recorrido empezó a las 12:27 y terminó a las 12:39, en este intervalo de tiempo se esperaba recibir un total de 13 mensajes Sigfox, pero solo se obtuvieron 12 mensajes Sigfox, es decir se recibió el 92.31% de los mensajes esperados.

HISTORIAL

Id dispositivo	Latitud	Longitud	Fecha
4D7226	-0.185538	-78.48556500000001	9/12/2021 12:27:17
4D7226	-0.185168	-78.486785	9/12/2021 12:28:20
4D7226	-0.190003	-78.485069	9/12/2021 12:30:26
4D7226	-0.188203	-78.483352	9/12/2021 12:31:29
4D7226	-0.184561	-78.48262000000001	9/12/2021 12:32:33
4D7226	-0.18167099999999997	-78.482185	9/12/2021 12:33:35
4D7226	-0.17928800000000003	-78.48176500000001	9/12/2021 12:34:39
4D7226	-0.176906	-78.482505	9/12/2021 12:35:40
4D7226	-0.178931	-78.48491600000001	9/12/2021 12:36:45
4D7226	-0.180948	-78.486343	9/12/2021 12:37:46
4D7226	-0.184603	-78.48661000000001	9/12/2021 12:38:50
4D7226	-0.18552	-78.485488	9/12/2021 12:39:52

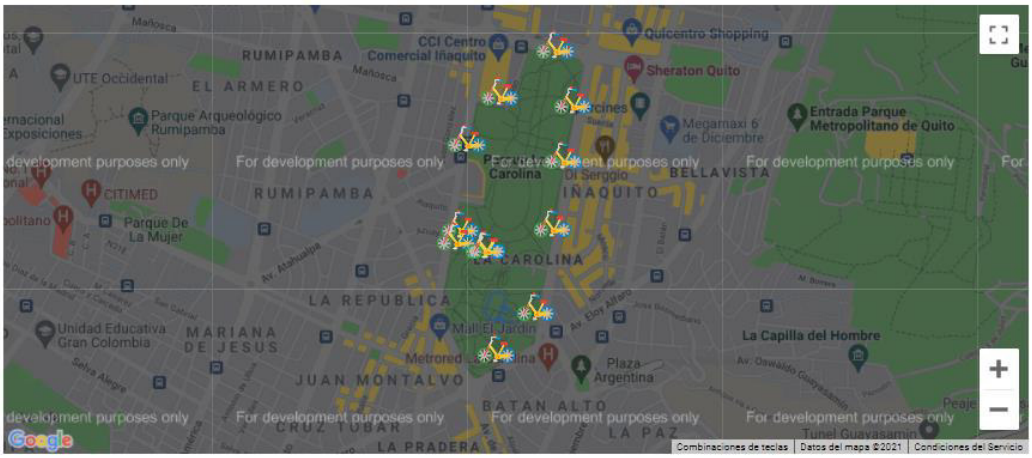


Figura 2.55 Historial del usuario "PARQUE CAROLINA 2" terminada la ruta.

2.3.2 PRUEBA 2 TUMBACO

La prueba dentro de Tumbaco se realizará un recorrido; el mapa de cobertura Sigfox dentro del parque se aprecia en la figura 2.56 y nos indica que no tenemos una cobertura de 3 estaciones base en todo el recorrido que se va a realizar en Tumbaco, tenemos zonas que tenemos 2 estaciones base, tenemos zonas en la que se tiene cobertura con una estación base e incluso tenemos zonas en donde no se tiene una cobertura de la red Sigfox.

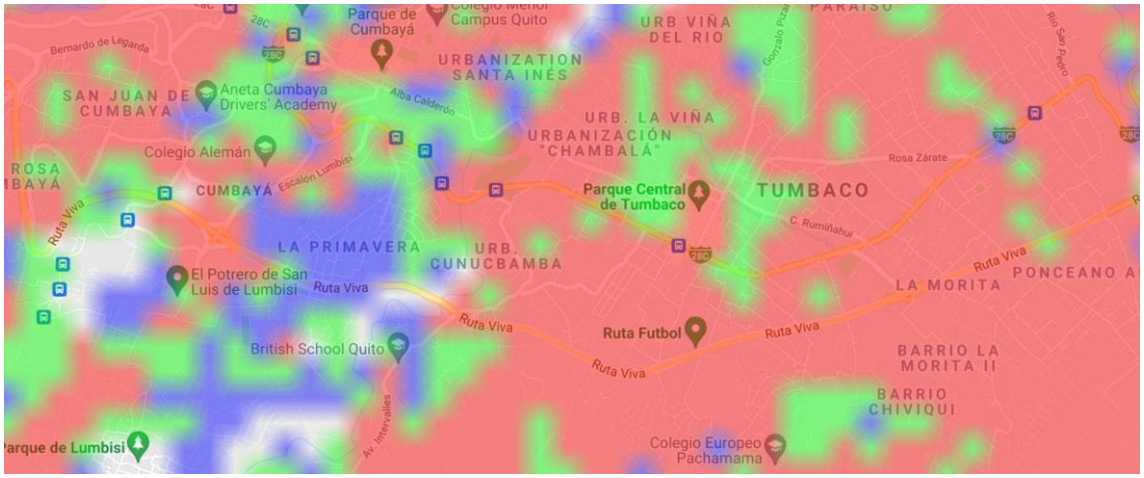


Figura 2.56. Mapa de cobertura recorrido Tumbaco.

2.3.2.1 Ruta dentro de Tumbaco

La ruta dentro de Tumbaco se encuentra descrito en la figura 2.57 que sale de Tumbaco en dirección a Puembo por medio la Avenida Oswaldo Guayasamín y regresamos por medio de la Avenida Guayasamín hasta la Plaza San Francisco en Cumbayá, el recorrido tiene una longitud aproximada de 13,5 kilómetros.

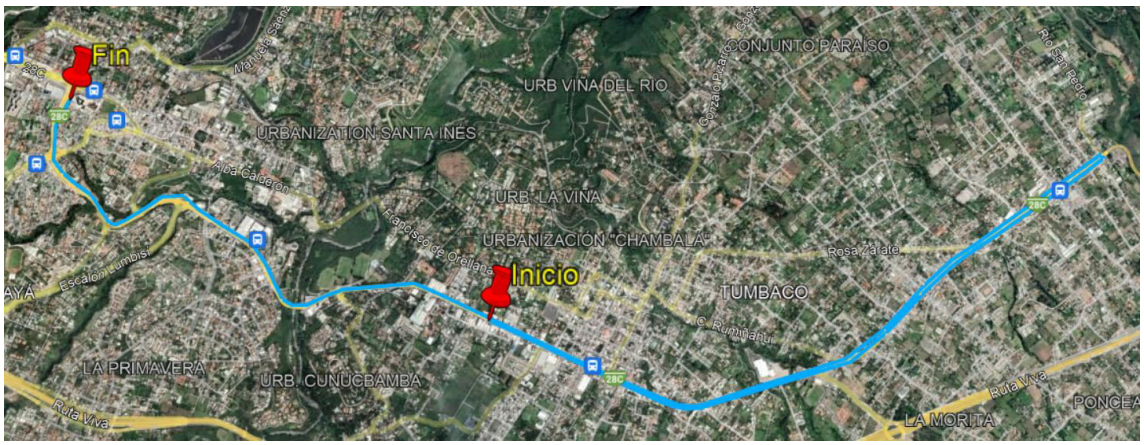


Figura 2.57. Ruta Tumbaco.

Para la prueba de Tumbaco se crea un usuario, tal y como se realizó para la prueba en el parque de la Carolina. Una vez creado el usuario se procede a seleccionar la bicicleta correspondiente y a iniciar el proceso de reserva de bicicleta; una vez finalizado la ruta procedemos a quitar la reserva correspondiente.

El historial obtenido en la prueba de Tumbaco se la puede apreciar en la figura 2.58.

HISTORIAL

Id dispositivo	Latitud	Longitud	Fecha
4D7226	-0.21231499999999998	-78.411827	29/11/2021 12:42:38
4D7226	-0.21734599999999998	-78.399208	29/11/2021 12:46:48
4D7226	-0.216028	-78.39488200000001	29/11/2021 12:48:54
4D7226	-0.21364599999999997	-78.38894599999999	29/11/2021 12:51:1
4D7226	-0.210424	-78.386001	29/11/2021 12:52:4
4D7226	-0.21714	-78.401252	29/11/2021 13:3:36
4D7226	-0.21636	-78.402961	29/11/2021 13:4:41
4D7226	-0.215068	-78.40576100000001	29/11/2021 13:5:41
4D7226	-0.21296299999999999	-78.410194	29/11/2021 13:7:47
4D7226	-0.211491	-78.41330699999999	29/11/2021 13:8:50
4D7226	-0.206606	-78.427864	29/11/2021 13:13:3
4D7226	-0.20508500000000002	-78.43170900000001	29/11/2021 13:15:9
4D7226	-0.199113	-78.437553	29/11/2021 13:19:22

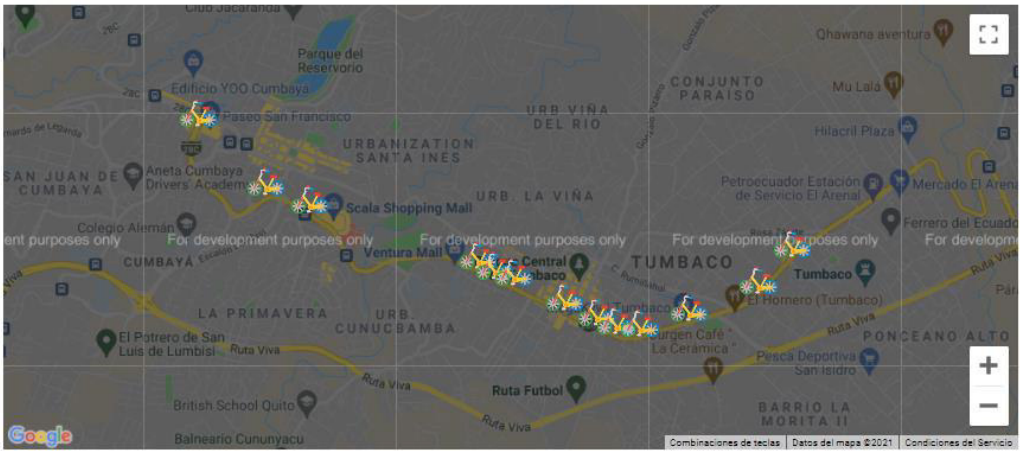


Figura 2.58. Historia de la ruta Tumbaco.

El recorrido empezó a las 12:42 y terminó a las 13:19, en este intervalo de tiempo se esperaba recibir un total de 38 mensajes Sigfox, pero solo se obtuvieron 13 mensajes Sigfox, es decir se recibió solo el 34.21% de los mensajes esperados.

2.3.3 PRUEBA 3 SANGOLQUÍ

La prueba dentro de Sangolquí se realizará un recorrido; el mapa de cobertura Sigfox dentro del parque se aprecia en la figura 2.59 y nos indica que no tenemos una cobertura de 3 estaciones base en todo el recorrido que se va a realizar en Sangolquí, tenemos zonas que tenemos 2 estaciones base, tenemos zonas en la que se tiene cobertura con una estación base e incluso tenemos zonas en donde no se tiene una cobertura de la red Sigfox.

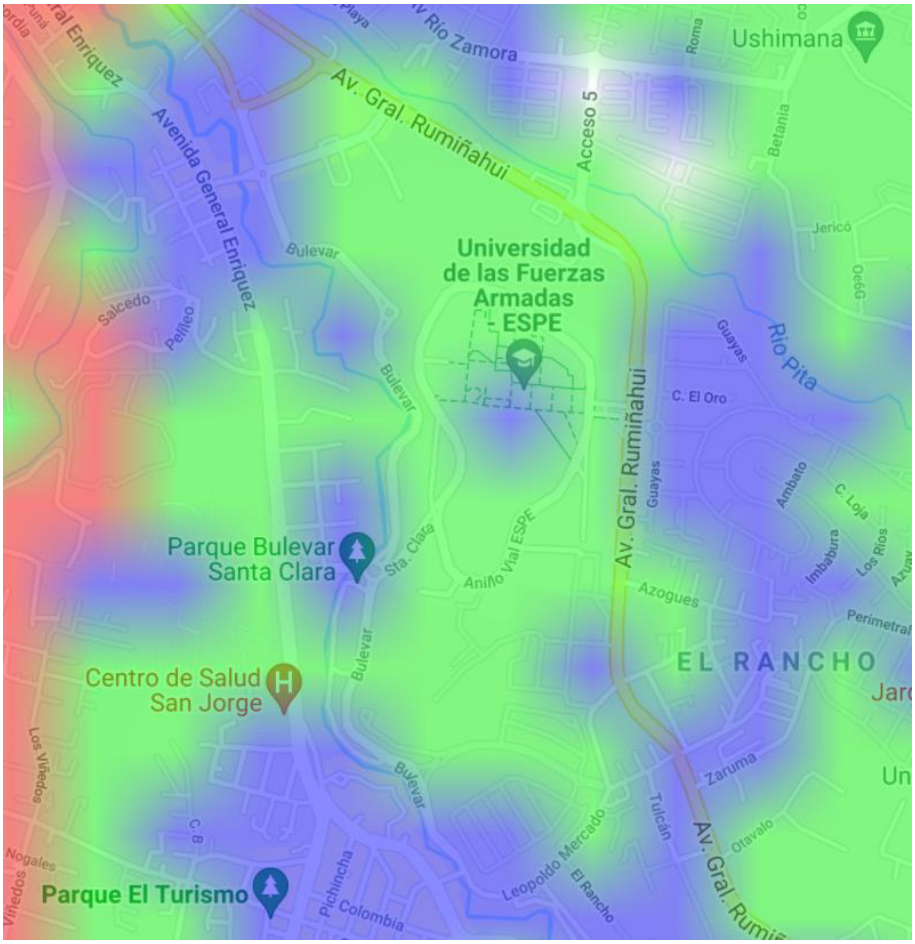


Figura 2.59. Mapa de cobertura recorrido Sangolquí.

2.3.3.1 Ruta dentro de Sangolquí.

La ruta dentro de Sangolquí se encuentra descrito en la figura 2.60 que sale del parqueadero del parque lineal Santa Clara en dirección al centro comercial San Luis Shopping por el bulevar Santa Clara y retorna al parqueadero por la avenida General Rumiñahui, el recorrido tiene una longitud aproximada de 5,5 kilómetros.

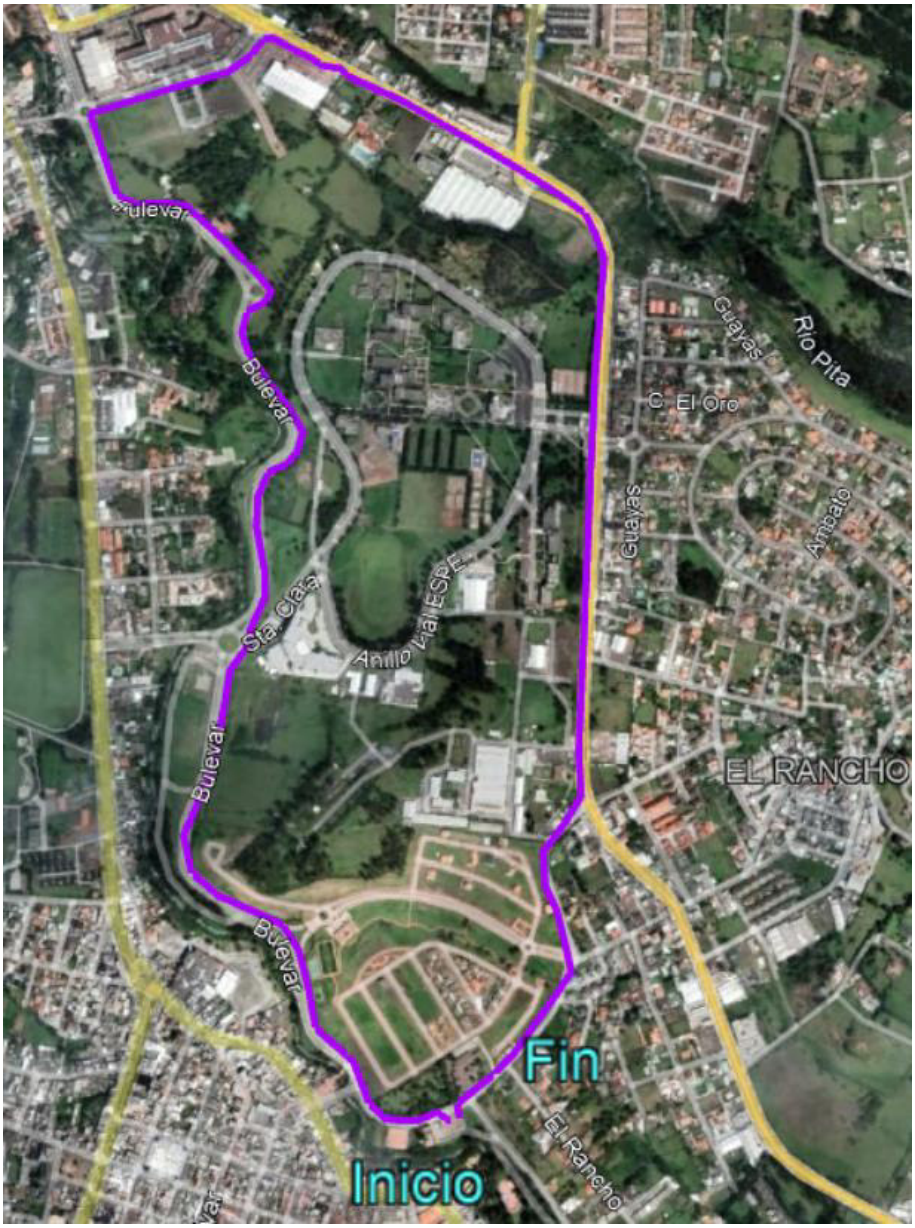


Figura 2.60. Ruta Sangolquí.

Para la prueba de Sangolquí se crea un usuario, tal y como se realizó para la prueba en el parque de la Carolina. Una vez creado el usuario se procede a seleccionar la bicicleta correspondiente y a iniciar el proceso de reserva de bicicleta; una vez finalizado la ruta procedemos a quitar la reserva correspondiente.

El historial obtenido en la prueba de Sangolquí se la puede apreciar en la figura 2.61.

HISTORIAL

Id dispositivo	Latitud	Longitud	Fecha
4D7226	-0.32645100000000005	-78.44456399999999	6/12/2021 10:49:25
4D7226	-0.326371	-78.444717	6/12/2021 10:50:28
4D7226	-0.322636	-78.448661	6/12/2021 10:52:36
4D7226	-0.318746	-78.44805099999999	6/12/2021 10:53:37
4D7226	-0.310941	-78.449066	6/12/2021 10:55:45
4D7226	-0.308855	-78.45010300000001	6/12/2021 10:56:46
4D7226	-0.31410099999999996	-78.44206199999999	6/12/2021 11:0:58
4D7226	-0.32433100000000004	-78.44281699999999	6/12/2021 11:4:7
4D7226	-0.326396	-78.444503	6/12/2021 11:5:9
4D7226	-0.32637600000000005	-78.44445	6/12/2021 11:6:12

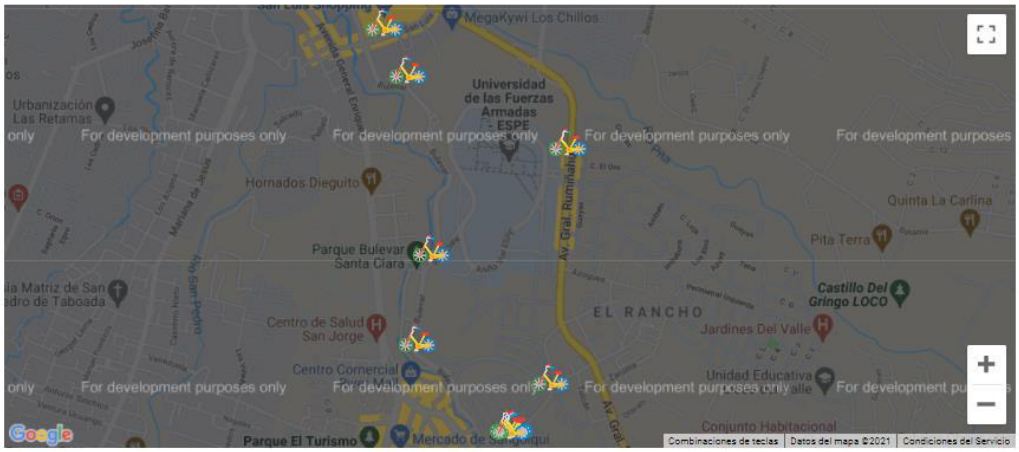


Figura 2.61. Historial de la ruta Sangolquí.

El recorrido empezó a las 10:49 y terminó a las 11:06, en este intervalo de tiempo se esperaba recibir un total de 18 mensajes Sigfox, pero solo se obtuvieron 10 mensajes Sigfox, es decir se recibió solo el 55.56% de los mensajes esperados.

3. RESULTADOS Y DISCUSIÓN

3.1 RESULTADOS

El prototipo de rastreo de bicicletas con tecnología Sigfox es posible implementarse mediante el esquema de la figura 2.2 y se encuentra limitado al área de cobertura Sigfox dentro del Ecuador.

Se realizaron 3 escenarios diferentes para probar el prototipo dentro de la provincia de Pichincha en el que se obtuvieron varios resultados.

En el escenario del parque de la Carolina se pudo verificar un adecuado funcionamiento de la cobertura Sigfox, ya que se obtuvo 92,31% de los mensajes esperados, con lo cual se pudo monitorear la ubicación de la bicicleta en su gran mayoría de tiempo.

En el escenario de Tumbaco se pudo verificar un inadecuado funcionamiento del prototipo ya que solo se obtuvo el 34,21% de los mensajes esperados, es decir que solo se pudo monitorear la ubicación de la bicicleta en la tercera parte del tiempo observado.

En el escenario de Sangolquí se pudo verificar un inadecuado funcionamiento del prototipo ya que solo obtuvo el 55,56% de los mensajes esperados, es decir que solo se pudo monitorear la ubicación de la bicicleta en un poca más de la mitad del tiempo observado.

La cobertura de la red Sigfox dentro de los 2 escenarios que no cumplen con un adecuado funcionamiento merma el rendimiento del prototipo, en el escenario en el que se cumple con un funcionamiento adecuado se tiene la cobertura total de su territorio con 3 o más estaciones base; debido a que la red Sigfox en los componentes de estaciones base de Sigfox y la Nube de Sigfox es propio del operador no es posible realizar un mayor análisis de la causa del inadecuado funcionamiento en los 2 escenarios.

4. CONCLUSIONES Y RECOMENDACIONES

4.1 CONCLUSIONES

El prototipo de rastreo de bicicletas con la red Sigfox permite gestionar el alquiler de bicicletas dentro del Distrito Metropolitano de Quito por medio de la red Sigfox, el trabajo en el prototipo nos brindó mucha experiencia en la red Sigfox y se puede concluir lo siguiente:

- La cobertura de la red Sigfox en el Distrito Metropolitano de Quito permite la implementación del prototipo en ciertos sectores donde se tiene la cobertura con 3 o más estaciones base Sigfox.
- La cobertura de la red Sigfox en el Ecuador no permite la implementación de proyectos con la red Sigfox a nivel nacional debido a la falta de cobertura, por el momento se cuenta con cobertura parcial en las provincias de Azuay, Carchi, Cotopaxi, Guayas, Imbabura, Loja, Pichincha, Tungurahua y Santa Elena.
- El costo de utilización de la red Sigfox es significativamente económica en relación con la tecnología celular y satelital. Si el proyecto requiere de una gran cantidad de dispositivos este resulta conveniente debido a que el costo de utilización de la red se reduce mientras más dispositivos se vincule al proyecto.
- El backend de la nube Sigfox permite la implementación de aplicaciones de rastreo debido a la funcionalidad de la API Callback y su integración con la nube AWS.
- El servicio IoT Core de la nube AWS permite el monitoreo de los mensajes que son recibidos de parte del backend de la nube Sigfox.
- El servicio Lambda de la nube AWS permite la ejecución de códigos sin la necesidad de tener un servidor activo, esto permite un mayor control de cualquier solicitud y la posibilidad de escalar sin necesidad de requerir mayores recursos computacionales.
- El acuerdo de nivel de suscripción que mantiene actualmente la red Sigfox no permite la implementación de un sistema de monitoreo, debido a su limitante de mensajes diarios.

4.2 RECOMENDACIONES

- Es posible utilizar la tecnología Sigfox permite realizar diferentes proyectos con un bajo costo de comunicación.
- Sería de gran utilidad que se logre una mayor cobertura de la red Sigfox dentro del territorio del Ecuador para facilitar la implantación de proyectos.

5. REFERENCIAS BIBLIOGRÁFICAS

[1] K. Ritesh, P. Priyesh, B. Rafael and W. Maarten, "Energy Consumption Analysis of LPWAN," *sensors,* p. 20, 2020.

[2] W. Abdallah, S. Mnasri, N. Nasri y T. Val, Emergent IoT Wireless Technologies beyond the year 2020: A Comprehensive Comparative Analysis, de *International Conference on Computing and Information Technology*, Kingdom of Saudi Arabia, 2020.

[3] J. Cotrim y J. Kleinschmidt, LoRaWAN mesh networks: A review and classification of multihop communication., *Sensors,* pp. 1-21, 2020.

[4] Sigfox, "Sigfox," [Online]. Available: https://www.sigfox.com/en. [Accessed 20 Junio 2021].

[5] Sigfox, Sigfox connected objects - radio specifications, Febrero 2020. [En línea]. Available: https://build.sigfox.com/sigfox-device-radio-specifications#community-feedback. [Último acceso: Junio 2021].

[6] Amazon, AWS, [En línea]. Available: https://aws.amazon.com/es/what-is-cloud-computing/?nc1=h_ls. [Último acceso: 20 Junio 2021].

[7] Amazon Web Services, AWS Lambda, [En línea]. Available: https://aws.amazon.com/es/lambda/. [Último acceso: 24 Junio 2021].

[8] Amazon Web Services, Amazon DynamoDB, [En línea]. Available: https://aws.amazon.com/es/dynamodb/. [Último acceso: 24 Junio 2021].

[9] Amazon Web Services, Amazon API Gateway, [En línea]. Available: https://aws.amazon.com/es/api-gateway/. [Último acceso: 24 Junio 2021].

[10] Amazon Web Services, AWS IoT Core, [En línea]. Available: https://aws.amazon.com/es/iot-core/. [Último acceso: 24 Junio 2021].

[11] Amazon Web Services, AWS Identity & Access Management, [En línea]. Available: https://aws.amazon.com/es/iam/. [Último acceso: 24 Junio 2021].

[12] Amazon Web Services, AWS CloudFormation [En línea]. Available: https://aws.amazon.com/es/cloudformation. [Último acceso: 28 Junio 2021].

[13] Pycom, Pycom.io 23 Junio 2020. [En línea]. Available: https://docs.pycom.io/gitbook/assets/PyTrack2X_specsheet.pdf. [Último acceso: 15 Mayo 2021].

[14] Pycom, Pycom.io 22 Marzo 2018. [En línea]. Available: https://pycom.io/wp-content/uploads/2018/08/sipy-specsheet.pdf. [Último acceso: 20 Mayo 2021].

[15] Y. Zheng y L. Mingquan, Next-Generation GNSS Signal Design, Beijing: Springer, 2016.

[16] EUSPA, Constellation Information [En línea]. Available: https://www.gsc-europa.eu/system-service-status/constellation-information. [Último acceso: 25 Junio 2021].

[17] IBM, Representational State Transfer [En línea]. Available: https://www.ibm.com/docs/en/rational-soft-arch/9.7.0?topic=style-representational-state-transfer. [Último acceso: 10 Julio 2021].

[18] Red Hat, REST vs. SOAP [En línea]. Available: https://www.redhat.com/es/topics/integration/whats-the-difference-between-soap-rest. [Último acceso: 10 Julio 2021].

[19] Python, What is Python? Executive Summary [En línea]. Available: https://www.python.org/doc/essays/blurb/. [Último acceso: 2021 Agosto 11].

[20] Micropython, Proper Python with hardware-specific modules [En línea]. Available: https://micropython.org/. [Último acceso: 11 Agosto 2021].

[21] G. G. L. Ribeiro, L. F. De Lima, L. Oliveira, J. J. P. C. Rodrigues, C. N. M. Marins y G. A. B. Marcondes, An outdoor localization system based on SigFox de *Vehicular Technology Conference*, Brazil, 2018.

ANEXO 1

MANUAL DE USUARIO DEL PROTOTIPO DE RASTREO DE BICICLETAS CON TECNOLOGÍA SIGFOX

Índice de Contenido

1. Manual de registro de dispositivo Sipy en la plataforma Sigfox

1.1 Actualización de Firmware

El dispositivo Sipy 1.0 de la marca Pycom cuenta con la posibilidad de comunicarse con la red Sigfox por medio de su chip CC1125 de la marca Texas Instruments. En la figura 1.1.1 se puede apreciar el dispositivo Sipy 1.0.

Figura 1.1.1. Dispositivo Sipy 1.0.

Para el registro del dispositivo Sipy 1.0 se necesita de una placa de desarrollo como la Pytrack 2.0 de la marca Pycom, en la figura 1.1.2 se puede apreciar el dispositivo Pytrack 2.0.

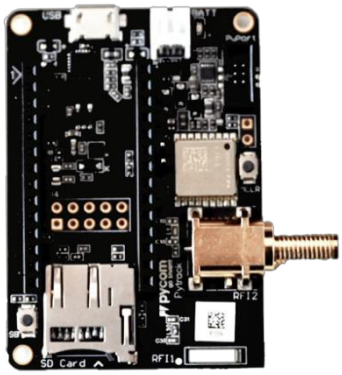


Figura 1.1.2. Dispositivo Pytrack 2.0.

Para iniciar el proceso de registro del dispositivo Sipy 1.0 se debe montar el dispositivo Sipy 1.0 en la placa de desarrollo Pytrack 2.0 como se aprecia en la figura 1.1.3.

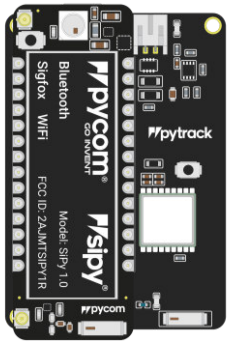


Figura 1.1.3. Dispositivo nodo Pycom.

Una vez se tenga listo el nodo Pycom que se encuentra conformado por los 2 dispositivos se debe actualizar el firmware del dispositivo Sipy 1.0 por medio del programa "Pycom Firmware Update" que se lo puede obtener en el link https://docs.pycom.io/updatefirmware/device/

Una vez instalado el programa se debe conectar el nodo Pycom por medio del cable USB al computador para actualizar el firmware, es necesario contar con conexión a internet para poder actualizar el firmware. Se debe seleccionar el puerto de comunicaciones asignado al dispositivo y escoger los parámetros de la figura 1.1.4 y damos clic en el botón siguiente.

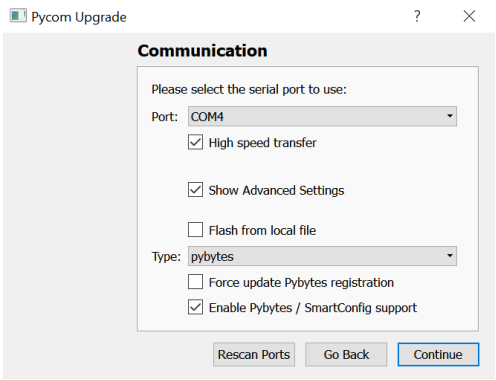


Figura 1.1.4. Parámetros de actualización de firmware.

En la ventana de configuraciones avanzadas se debe completar con los parámetros descritos en la figura 1.1.5 y damos clic en siguiente.

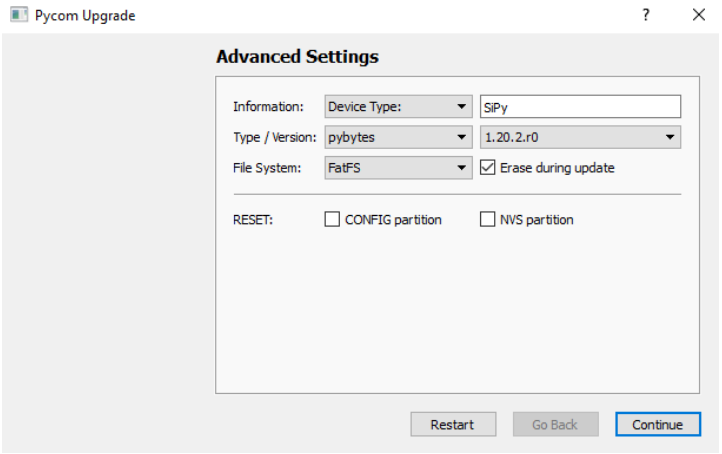


Figura 1.1.5. Parámetros de configuración avanzada.

Una vez finalizada la actualización del firmware del equipo Sipy 1.0 el programa nos mostrará información correspondiente a el Sigfox ID y el Sigfox PAC que será de gran utilidad en el registro del dispositivo en el portal de Sigfox. En la figura 1.1.6 se muestra un ejemplo del resultado exitoso de la actualización del dispositivo Sipy 1.0.

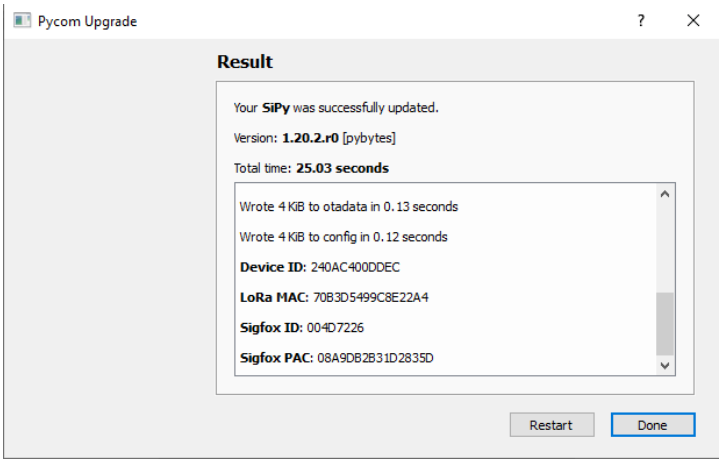


Figura 1.1.6. Resultado de la actualización del dispositivo Sipy 1.0.

1.2 Registro dispositivo Sipy 1.0 en el backend Sigfox.

Una vez obtenido los parámetros Sigfox ID y Sigfox PAC del paso anterior se procede con el registro del dispositivo en el backend de Sigfox. Nos dirigimos al link https://backend.sigfox.com/auth/login y procedemos a autenticarnos con las credenciales de nuestra cuenta creada en el link https://build.sigfox.com/sign-up.

Una vez realizado la autenticación nos aparecerá el portal de backend de Sigfox tal como se aprecia en la figura 1.2.1.

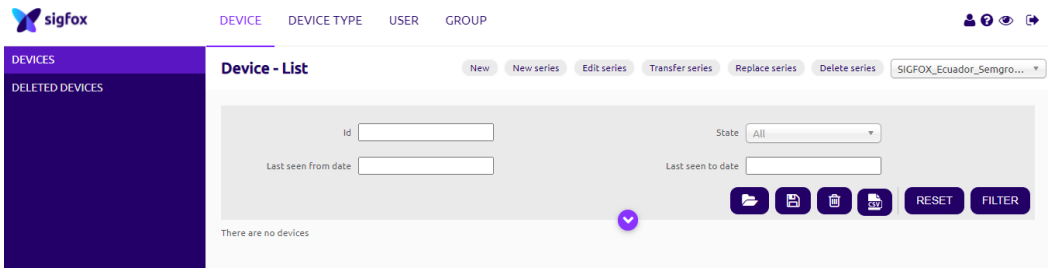


Figura 1.2.1. Portal de la nube de Sigfox.

Para registrar el nuevo dispositivo nos debemos colocar en la pestaña “DEVICE” y dar clic en “New”, y completar la información de la figura 1.2.2.

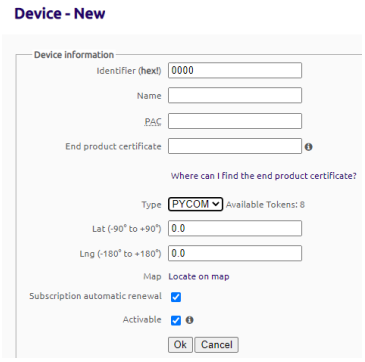


Figura 1.2.2. Información del nuevo dispositivo.

Una vez registrado el dispositivo se lo podrá visualizar en la pestaña “DEVICE” tal y como se aprecia en la figura 1.2.3.

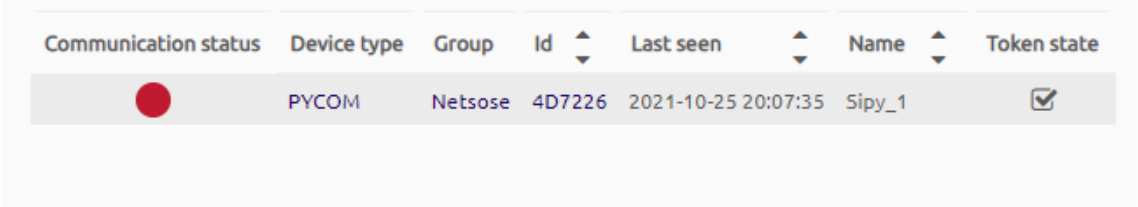


Figura 1.2.3. Registro de dispositivo Sipy 1.0**.**

2. Manual de instalación de Software Atom y Pymakr

2.1 Instalación Atom

El software Atom es un editor de código fuente de código abierto para macOS, Linux, y Windows con soporte para control de versiones Git integrado, desarrollado por GitHub. Atom es una aplicación de escritorio construida utilizando tecnologías web. El software es de libre uso y se lo puede obtener del link https://atom.io/

Una vez realizada la instalación del software Atom se puede proceder a la instalación del complemento Pymakr.

2.2 Instalación complemento Pymakr

La instalación del complemento Pymakr se debe seguir los siguientes pasos:

1. Seleccionar la pestaña “File” en el software Atom.
2. Seleccionar la opción “Settings” de la pestaña “File”.
3. Nos dirigimos a la opción de “Packages” y buscamos el complemento “Pymakr”
4. Una vez encontrado el complemento procedemos con la instalación de este, al finalizar la instalación se debe apreciar el resultado como la figura 2.2.1.

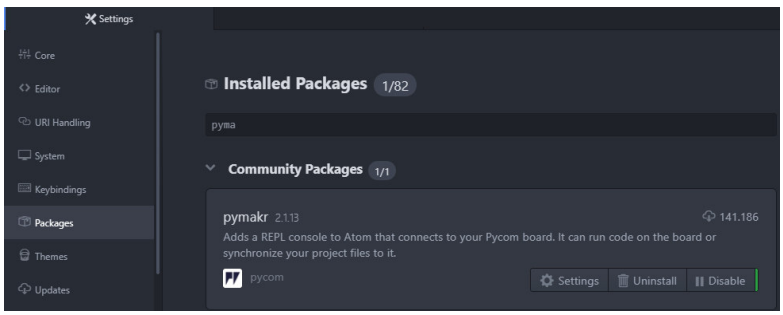


Figura 6. Resultado de la instalación de Pymakr.

3. Manual de carga del código del Software Atom al nodo Pycom.

Para cargar el código que se ejecutará dentro del nodo Pycom es necesario tener en cuenta las librerías necesarias para ejecutar el código principal y todos estos archivos se deben cargar dentro del nodo Pycom. Para el "PROTOTIPO DE RASTREO DE BICICLETAS CON TECNOLOGÍA SIGFOX" se utilizaron varias librerías de código abierto de la empresa Pycom y el código principal desarrollado por el autor.

Todos los archivos se encuentran organizados en una carpeta, que deberá ser abierta por el software Atom.

Una vez abierto el proyecto en el software Atom procedemos a conectar el nodo Pycom por medio de cable USB al computador y esperamos a que el complemento Pymakr lo reconozca en su interfaz, tal y como se aprecia en la figura 3.1

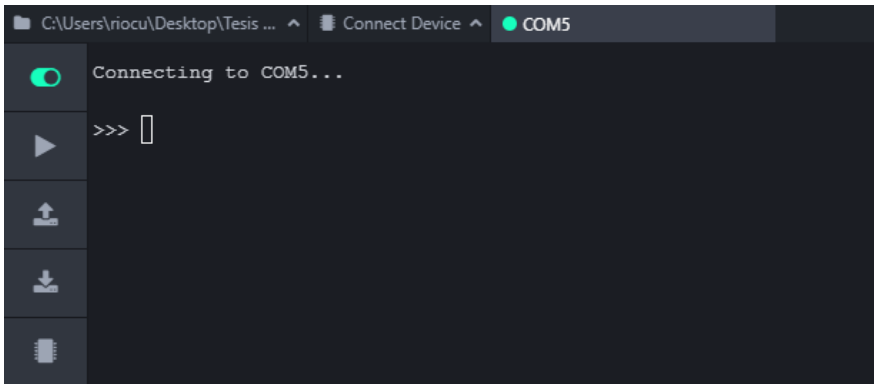


Figura 3.1. Software Atom con Pymakr

Una vez vinculado el nodo Pycom con el software Atom se procede a subir el programa al nodo Pycom, tal como se aprecia en la figura 3.2.

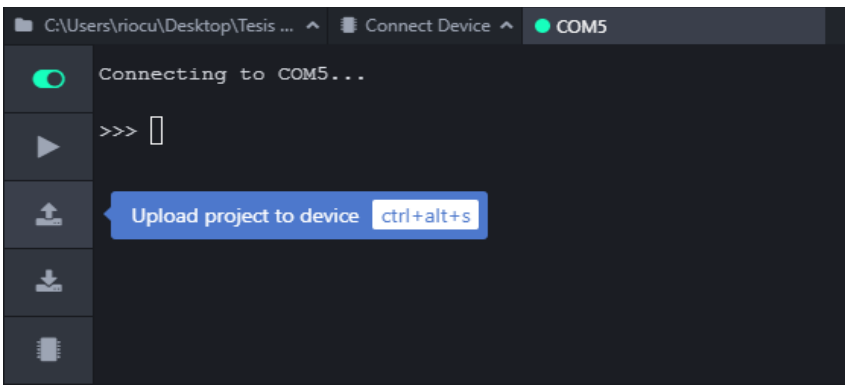


Figura 3.2. Carga del programa al Nodo Pycom.

Una vez finalizada la carga de los archivos y si no se obtuvo ningún error nos debe aparecer el resultado de la figura 3.3.

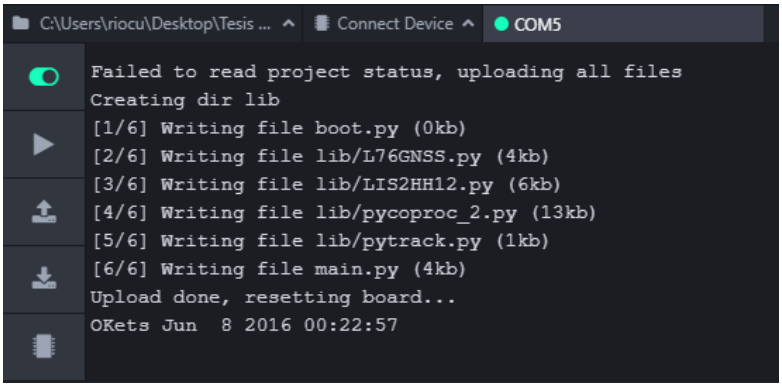


Figura 3.3. Carga exitosa del programa al Nodo Pycom.

Una vez realizada la carga del programa al nodo Pycom se puede apreciar la llegada de los mensajes al backend de Sigfox en su sección de “Devices” en la opción de mensajes, tal como se aprecia en la figura 3.4.

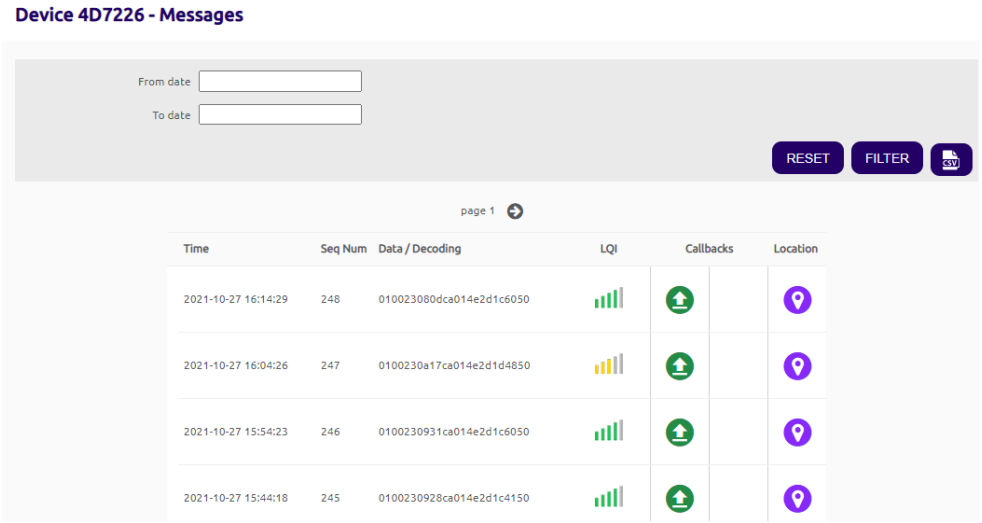


Figura 3.4. Mensajes del nodo Pycom en el backend de Sigfox.

4. Manual de uso del Aplicativo del prototipo.

El aplicativo del prototipo permite el registro de nuevos usuarios, la autenticación de usuarios al aplicativo, la reserva de una bicicleta disponible por medio de la visualización de la posición GPS del nodo Pycom en un mapa, quitar la reserva y el registro automático del historial de uso de cada usuario.

4.1 Registro de nuevo usuario.

Para el registro de un nuevo usuario en el aplicativo es necesario llenar los campos de la forma de la figura 4.1.1.

Figura 4.1.1. Forma de registro de nuevos usuarios.

Una vez llenado todos los campos, se debe hacer clic en “Enviar”.

Un ejemplo de un usuario creado se lo aprecia en la figura 4.1.2.

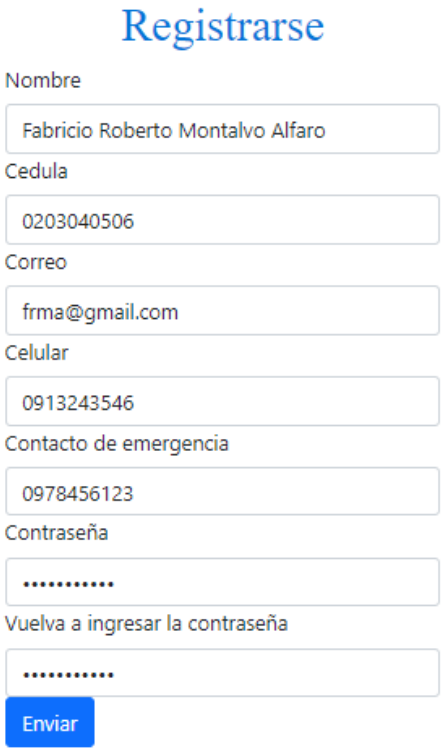


Figura 4.1.2. Ejemplo de registro de usuario.

4.2 Autenticación de usuario.

Una vez se ha registrado el usuario en el aplicativo, se puede autenticar para hacer uso de este, para esto debe dar clic en la pestaña de "Inicio de Sesión" e ingresar sus credenciales.

En el caso de ingresar incorrectamente la contraseña obtendrá el mensaje de "Contraseña incorrecta"

En el caso de ingresar incorrectamente el número de cédula, obtendrá el mensaje de "El usuario no existe"

En el caso de ingresar exitosamente el número de cédula y la contraseña obtendrá acceso al aplicativo donde se mostrará 3 botones como se muestra en la figura 4.2.1.

Figura 4.2.1. Opciones del aplicativo.

4.3 Reserva de bicicleta disponible.

Al dar clic en el botón “Reservar” se abre una página donde se aprecian las bicicletas disponibles para la reserva, como se aprecia en la figura 4.3.1.

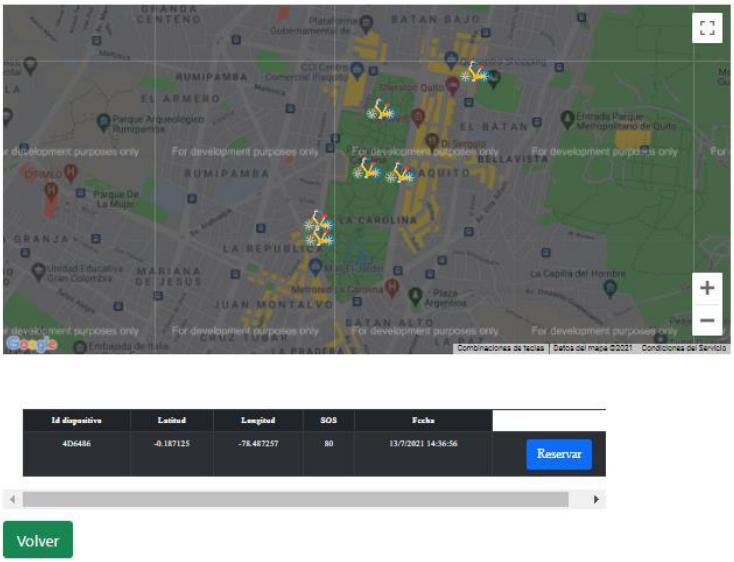


Figura 4.3.1. Página para reserva de bicicleta.

Se debe seleccionar una bicicleta que se encuentre dentro del mapa y se mostrará información de su estatus, usted tiene la opción de reservarla o buscar otra bicicleta.

Si reserva la bicicleta, obtendrá un mensaje de “reserva exitosa” y se empezará a registrar su uso en el historial del perfil del usuario.

Es necesario conocer que solo se puede tener una reserva activa por cada usuario.

4.4 Quitar reserva.

Al dar clic en “Quitar reserva”, obtendrá un mensaje de “reserva finalizada” y se quita la reserva de la bicicleta que se encuentre activa para el usuario y se terminará el registro de uso en el historial del perfil de usuario.

4.5 Historial de uso de la bicicleta.

Al dar clic en “Historial”, obtendrá en una tabla el historial de uso del aplicativo, precisando la ubicación, el id del dispositivo utilizado, la fecha y la hora. Un ejemplo de esta tabla se la puede apreciar en la figura 4.5.1.

HISTORIAL

Id dispositivo	Latitud	Longitud	Fecha
4D7226	-0.32645100000000005	-78.44456399999999	6/12/2021 10:49:25
4D7226	-0.326371	-78.444717	6/12/2021 10:50:28
4D7226	-0.322636	-78.448661	6/12/2021 10:52:36
4D7226	-0.318746	-78.44805099999999	6/12/2021 10:53:37
4D7226	-0.310941	-78.449066	6/12/2021 10:55:45
4D7226	-0.308855	-78.45010300000001	6/12/2021 10:56:46
4D7226	-0.31410099999999996	-78.44206199999999	6/12/2021 11:0:58
4D7226	-0.32433100000000004	-78.44281699999999	6/12/2021 11:4:7
4D7226	-0.326396	-78.444503	6/12/2021 11:5:9
4D7226	-0.32637600000000005	-78.44445	6/12/2021 11:6:12

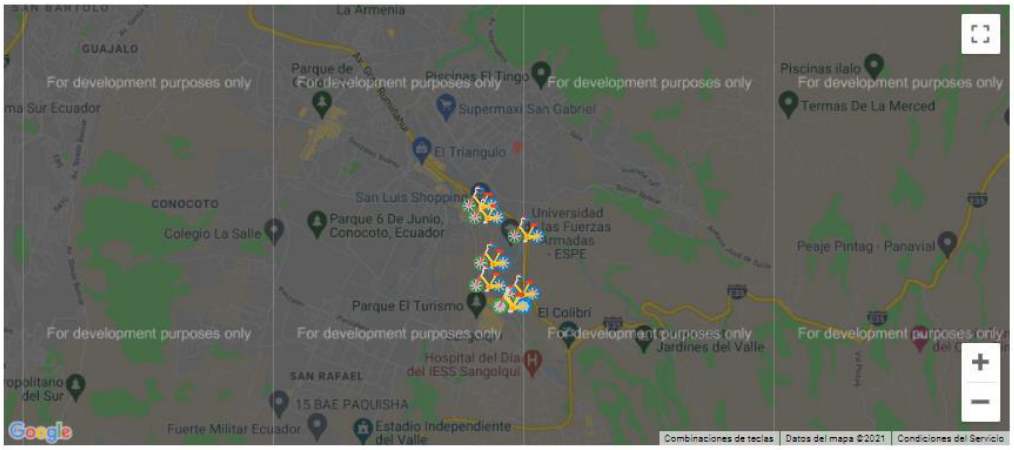


Figura 4.5.1. Ejemplo del historial de uso.

ORDEN DE EMPASTADO