DockerTraccar - Geofences

I. Resumen

***Abstract*—Se aborda la implementación y uso de la API de geocercas (Geofences) en el sistema de rastreo GPS Traccar. Se detalla cómo configurar y utilizar la API para gestionar geocercas a través de operaciones CRUD (Crear, Leer, Actualizar, Eliminar), utilizando herramientas como Docker para aislar y desplegar el entorno de desarrollo, se analiza la viabilidad de dockerizar la API, destacando los beneficios de consistencia, escalabilidad y portabilidad que ofrece Docker en la gestión y operación de esta API. Se incluyen instrucciones para configurar el entorno, generar tokens de autenticación y ejecutar comandos para interactuar con la API de geocercas de manera efectiva.**

II. Introducción

Este informe presenta una documentación detallada de la API de geocercas (Geofences) de Traccar, pero primeramente ¿que es traccar? un sistema de rastreo GPS de código abierto que permite la gestión y monitoreo de dispositivos en tiempo real. Traccar es ampliamente utilizado para rastrear vehículos, personas y objetos, ofreciendo una plataforma robusta para la gestión de flotas y la seguridad personal.

Fig. 1. Traccar

La API de Geofences de Traccar es utilizada para gestionar las geocercas (geofences en inglés) dentro del sistema Traccar. Una geocerca es un perímetro virtual definido por coordenadas geográficas que puede asociarse a un dispositivo de rastreo para monitorear si este entra o sale de un área específica [1]. En este documento se explorarán las funcionalidades, configuraciones y mejores prácticas para la implementación efectiva de geocercas utilizando Traccar. La API Geofences desempeña los siguientres roles [2]:

1) **Crear Geocercas:** Puedes definir nuevas geocercas proporcionando las coordenadas que delimitan el área de interés.
2) **Actualizar Geocercas:** Permite modificar los detalles de una geocerca existente, como su forma, nombre o las coordenadas.
3) **Eliminar Geocercas:** Permite eliminar una geocerca específica del sistema.
4) **Obtener Geocercas:** Puedes recuperar la lista de geocercas definidas en el sistema, junto con sus detalles (como las coordenadas, nombre, etc.).
5) **Asociar Geocercas a Dispositivos o Grupos:** Permite asociar una geocerca específica a uno o más dispositivos o grupos de dispositivos, para que estos sean monitoreados en relación con esa geocerca.

III. Materiales y Métodos

Para realizar este estudio, primeramente debemos tener en cuenta los materiales y recursos a utilizar como:

Computadora 1

- **Nombre del Dispositivo:** Laptop DELL DESKTOP
- **Procesador**: Intel Core i5-1135G7 8M Cache, hasta 4.20 GHz.
- **Procesador de Texto:** LATEX.
- **Herramienta:** Overleaf
- **Acceso a Internet**
- **Máquina Virtual:** VirtualBox - Docker Composer
- **ISO:** Debian
- **Bases de Datos Académicas:** Google Scholar - IEEE - SCOPUS
- **Libros y Artículos Académicos**

Computadora 2

- **Nombre del Dispositivo:** Laptop Dell Inspiron (DESKTOP-GT86D3N)
- **Procesador:** 11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz 2.30 GHz

- **RAM instalada:** 8.0 GB
- **Maquina Virtual:** VirtualBox - Docker Composer
- **ISO:** Debian
- **Conexión a Internet**

IV. Metodología

1. INTRODUCCIÓN A LA API/ENDPOINT GEOFENCES

Se desglosa los métodos con los cuales trabaja la API Geofences, los cuales son 4 y se podría decir que son los más comunes [3]:

1) **GET:** Este método se utiliza para obtener una lista de todas las geocercas definidas en el sistema Traccar. Cuando se realiza una solicitud GET a esta API, devuelve una colección de objetos de geocercas con sus detalles, .

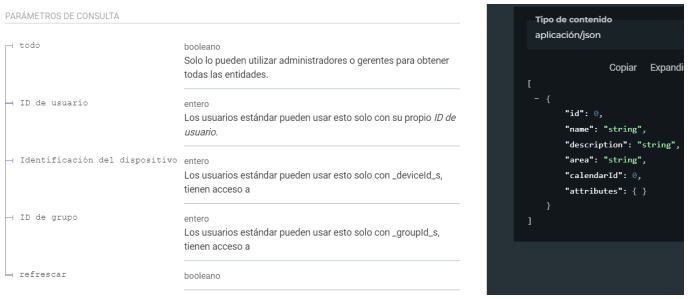

Fig. 2. GET

2) **POST:** Este método permite crear una nueva geocerca en Traccar. Al enviar una solicitud POST con los datos necesarios (por ejemplo, las coordenadas que definen la geocerca y otros atributos), se crea una nueva geocerca en el sistema.

Fig. 3. POST

3) **PUT:** Con este método, se puede actualizar una geocerca existente. Se utiliza para modificar las propiedades de una geocerca específica, como cambiar sus límites o cualquier otro atributo relevante. La solicitud PUT debe incluir el identificador de la geocerca a modificar junto con los nuevos datos.

Fig. 4. PUT

4) **DEL:** Este método se usa para eliminar una geocerca específica del sistema Traccar. Al enviar una solicitud DELETE con el identificador de la geocerca, se elimina esa geocerca del sistema.

Fig. 5. PUT

Para determinar si la API de Geofences de Traccar es un buen candidato para ser dockerizada, es necesario considerar varios factores relacionados con la arquitectura, dependencias, y operación de esta API dentro del sistema Traccar. Pero segun el analisis hecho la API de Geofences de Traccar es un buen candidato para ser dockerizada. Los beneficios de la consistencia del entorno, la facilidad de despliegue, la escalabilidad, y la portabilidad hacen que Docker sea una herramienta adecuada para gestionar y operar esta API de manera eficiente. Pero es ideal realizar pruebas para una eficiente dockerización.

2. DOCUMENTACIÓN DE LA API

Este documento proporciona una guía paso a paso sobre cómo configurar y utilizar la API de Geofences en el proyecto DockerTraccar. Se explicará cómo interactuar con la API utilizando comandos en PowerShell para realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre las geo-zonas.

1) **Prerrequisitos**
   Antes de comenzar, asegúrate de tener lo siguiente:
   - Docker y Docker Compose instalados y en ejecución.
   - Un editor de texto o entorno de desarrollo integrado (IDE), como Visual Studio Code (VSC).
   - Acceso al proyecto DockerTraccar
   - Conocimiento básico de PowerShell o terminales de línea de comandos.
2) **Configuración del Entorno**
   Abre el proyecto en VSC y realiza lo siguiente:
   - Inicia los Contenedores Docker

- Abre la terminal en VSC (Ctrl +) y navega al directorio del proyecto donde se encuentra el archivo docker-compose.yml.
- Ejecuta el siguiente comando *docker-compose up -d* para iniciar los contenedores:

3) **Verificación del estado de DockerTraccar**
   - El servidor Traccar debería estar disponible en http://localhost:8082.
   - Puedes verificarlo abriendo un navegador web e ingresando la URL.
4) **Generación del Token de Autenticación**
   Para interactuar con la API de Traccar, necesitarás un token de autenticación en base64.
   - Abre una terminal Bash en VSC.
   - Ejecuta el siguiente comando *echo -n "tu usuario:tu contraseña" | base64* para codificar tus credenciales (username:password) en base64
     Ejemplo:

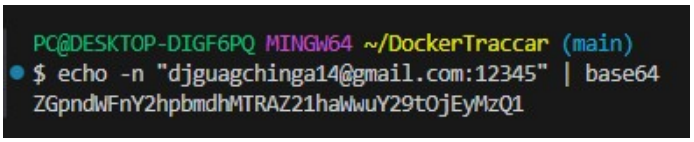

Fig. 6. Token de Autenticación

5) **Realización de Operaciones CRUD con la API de Geofences**
   NOTA:
   - Para realizar las operaciones CRUD de la API Geofences puedes utilizar la terminal de comandos: **Powershell o Bash de VSC**, en este caso utilizaremos **Bash**, ten en cuenta esto a la hora de ejecutar los comandos.
   - Adicional, en los comandos debes cambiar **tu-token-base64** por tu token de autenticación.

   A continuación los comandos para cada operación:

   a) **GET (Obtener Geofences)**
   Para obtener la lista de geofences existentes ejecuta en la terminal el siguiente comando:

```
curl -X GET "http://localhost
    :8082/api/geofences" \
-H "Authorization: Basic
    tu_token_base64"
```

EJEMPLO:

Fig. 7. Comando GET

Verificamos tanto en la web como en la API.

Fig. 8. API Geofences-GET

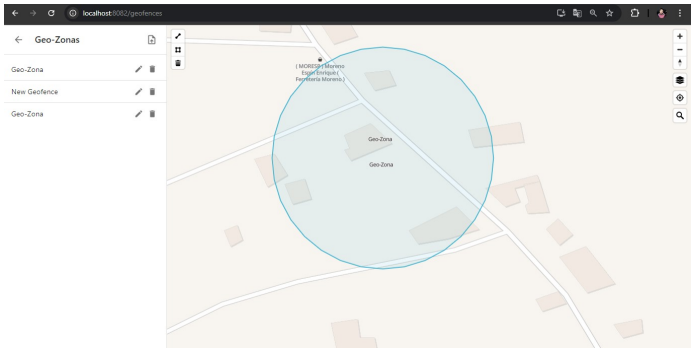

Fig. 9. Web Geofences

b) **POST (Crear una Nueva Geofence)**
Para crear una nueva geofence ejecuta el siguiente comando:

```
curl -X POST "http://localhost
    :8082/api/geofences" \
-H "Authorization: Basic
    tu_token_base64" \
-H "Content-Type: application/json
    " \
-d '{
    "name": "New Geofence",
    "area": "CIRCLE (1234 5678,
    300)"
}'
```

Puedes cambiar los datos según tus necesidades y preferencias.
EJEMPLO:

Fig. 10. Comando POST

Visualizamos los cambios tanto en la API como en la Web.

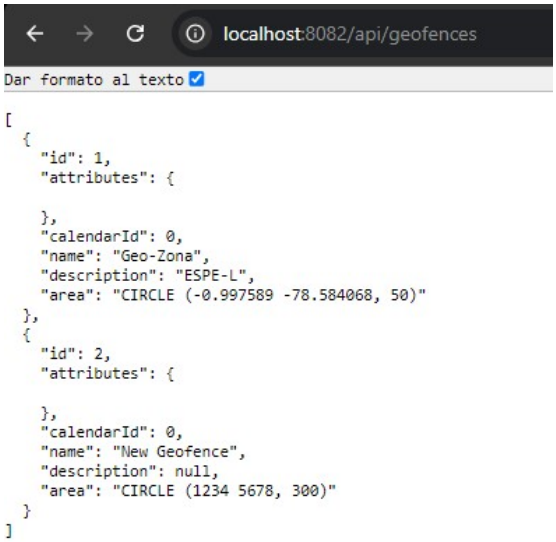

Fig. 11. API Geofences-POST

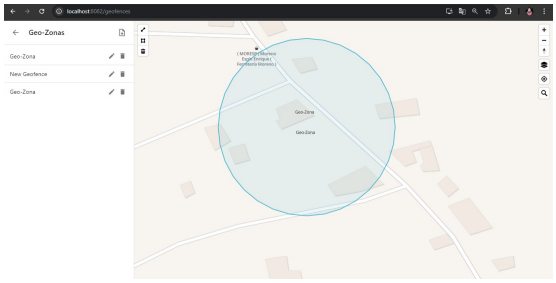

Fig. 12. Web Geofences-POST

c) **PUT (Actualizar una Geofence Existente)**
Para actualizar lo hacemos mediante el ID con el siguiente comando:

```
curl -X PUT "http://localhost
    :8082/api/geofences/<
    geofence_id>" \
-H "Authorization: Basic
    tu_token_base64" \
-H "Content-Type: application/json
    " \
-d '{
     "name": "Updated Geofence",
     "area": "CIRCLE (1234 5678,
    500)"
}'
```

Reemplaza <geofence-id> con el ID del geofence que deseas actualizar, en este caso ID: 2.
EJEMPLO:

Fig. 13. Comando PUT

Visualizamos los cambios tanto en la API como en la Web.

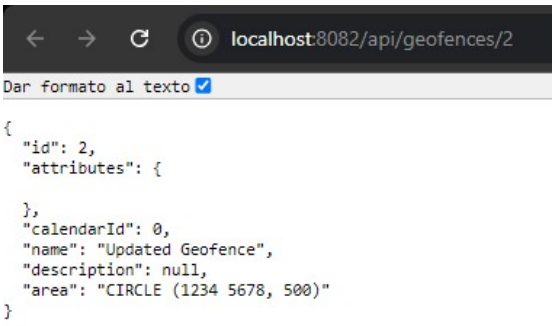

Fig. 14. API Geofences - PUT

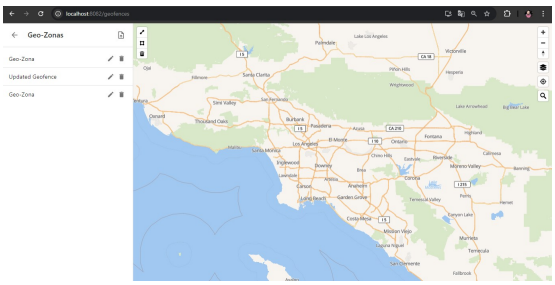

Fig. 15. Web Geofences - PUT

d) **DELETE (Eliminar una Geofence)**
Para eliminar lo hacemos mediante el ID con el siguiente comando:

```
curl -X DELETE "http://localhost
    :8082/api/geofences/<
    geofence_id>" \
-H "Authorization: Basic
    tu_token_base64"
```

Reemplaza <geofence-id> con el ID del geofence que deseas eliminar, en este caso: ID: 2.
EJEMPLO

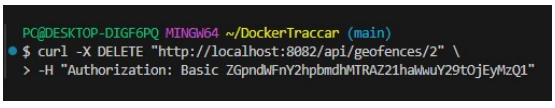

Fig. 16. Comando DELETE

Comprobamos que se ha eliminado la geofence con ID: 2 tanto en la API como en la web

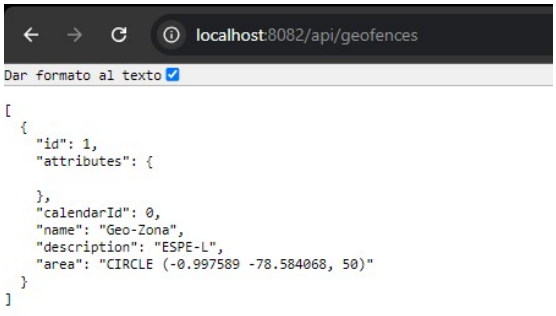

Fig. 17. API Geofences-DELETE

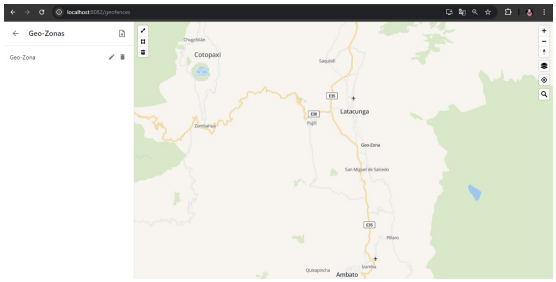

Fig. 18. Web Geofences-DELETE

3. DOCUMENTACIÓN DE LOS IMPORTS

A continuación se van a explicar los imports necesarios para la API Geofences:

IMPORT 1

```
import org.traccar.api.
   ExtendedObjectResource"
```

Descripción:
ExtendedObjectResource es parte del paquete org.traccar.api, que forma parte de la API del sistema Traccar, un sistema de rastreo de GPS de código abierto. Este recurso se utiliza principalmente para gestionar la comunicación y manipulación de objetos a través de la API, proporcionando operaciones extendidas que pueden no estar disponibles en los recursos básicos.
Cómo funciona:
Este recurso extiende la funcionalidad de recursos de objetos básicos en la API de Traccar, permitiendo la manipulación avanzada de objetos como dispositivos, usuarios, o geocercas a través de endpoints REST. Proporciona métodos adicionales para manejar operaciones complejas que requieren lógica personalizada, como validaciones adicionales o la aplicación de reglas de negocio antes de realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar).
Funcionalidades principales:

- Manipulación extendida de objetos: Permite realizar operaciones avanzadas sobre objetos que requieren lógica adicional más allá de las operaciones CRUD estándar.
- Validaciones personalizadas: Antes de realizar una operación, se pueden aplicar validaciones específicas o reglas de negocio para asegurar la consistencia de los datos.
- Interacción con otros componentes: Puede interactuar con otros recursos y servicios del sistema Traccar, permitiendo la ejecución de acciones en cadena o complejas.

Dependencias:

- org.traccar.api.BaseResource: ExtendedObjectResource suele extender clases básicas de recursos como BaseResource, que proporcionan las funcionalidades CRUD básicas.
- Servicios del sistema Traccar: Puede depender de varios servicios del sistema Traccar, como los servicios de validación, almacenamiento y lógica de negocio.
- Dependencias RESTful: Depende del framework de JAX-RS (Java API for RESTful Web Services) para manejar las solicitudes HTTP y realizar operaciones sobre los recursos.

Ejemplo de uso de ExtendedObjectResource
Este ejemplo muestra cómo podrías extender la funcionalidad de ExtendedObjectResource para manejar un objeto de dispositivo personalizado con validaciones adicionales antes de crear un nuevo dispositivo.

```
import org.traccar.api.
   ExtendedObjectResource;
import org.traccar.model.Device;
import org.traccar.database.DeviceManager
   ;

import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.core.Response;

@Path("custom/devices")
public class CustomDeviceResource extends
    ExtendedObjectResource<Device> {

    private final DeviceManager
   deviceManager;

    public CustomDeviceResource(
   DeviceManager deviceManager) {
        super(Device.class);
        this.deviceManager =
   deviceManager;
    }

    @POST
    public Response addDevice(Device
   device) throws Exception {
        // Validaci n personalizada
        if (device.getName() == null ||
   device.getName().isEmpty()) {
```

```
            return Response.status(
    Response.Status.BAD_REQUEST)
                            .entity("
    Device name cannot be empty")
                            .build();
        }

        // L gica adicional (por ejemplo
    , chequeo de duplicados)
        if (deviceManager.getByName(
    device.getName()) != null) {
            return Response.status(
    Response.Status.CONFLICT)
                            .entity("A
    device with this name already exists")
                            .build();
        }

        // Crear el dispositivo
        return super.add(device);
    }
}
```

Explicación:

- Validación personalizada: Se verifica que el nombre del dispositivo no esté vacío.
- Chequeo de duplicados: Se verifica si ya existe un dispositivo con el mismo nombre.
- Llamada a super.add(device);: Si las validaciones pasan, se utiliza la funcionalidad base de ExtendedObjectResource para agregar el dispositivo.

IMPORT 2

```
import org.traccar.model.Geofence;
```

Descripción:
Geofence es una clase dentro del paquete org.traccar.model, que representa un área geográfica virtual definida en el sistema Traccar. Las geocercas se utilizan para definir zonas específicas donde se rastrea si un dispositivo entra o sale.
Como funciona:
La clase Geofence modela una geocerca utilizando parámetros como coordenadas, tipo de forma (círculo, polígono), y atributos adicionales como el nombre y descripción. Los objetos de esta clase son instancias de geocercas que pueden asociarse a dispositivos, permitiendo generar alertas o eventos cuando un dispositivo cruza los límites de la geocerca.
Funcionalidades principales:

- Definición de zonas geográficas: Permite definir geocercas de diferentes formas (círculo, polígono) en un mapa.
- Asociación con dispositivos: Las geocercas se pueden asociar a uno o varios dispositivos para monitorear su entrada o salida de estas zonas.
- Generación de eventos: Cuando un dispositivo entra o sale de una geocerca, se generan eventos que pueden ser utilizados para notificaciones o acciones automáticas.
- Personalización de geocercas: Se pueden definir parámetros personalizados como nombre, descripción, y otros atributos relevantes para la gestión de geocercas.

Dependencias:

- org.traccar.model.BaseModel: Geofence usualmente extiende de BaseModel, lo que le proporciona funcionalidades comunes como el manejo de IDs, timestamps, y otras propiedades básicas de los modelos en Traccar.
- Servicios de almacenamiento: Depende de los servicios de almacenamiento del sistema Traccar para guardar, actualizar, y recuperar las definiciones de geocercas.
- Componentes de mapas y coordenadas: Utiliza componentes que manejan las coordenadas geográficas y cálculos de distancias para definir y trabajar con las geocercas.

Ejemplo de uso de Geofence Este ejemplo muestra cómo puedes crear y asociar una geocerca a un dispositivo en el sistema Traccar.

```
import org.traccar.model.Geofence;
import org.traccar.model.Device;
import org.traccar.database.
   GeofenceManager;
import org.traccar.database.DeviceManager
   ;

public class GeofenceExample {

    private GeofenceManager
   geofenceManager;
    private DeviceManager deviceManager;

    public GeofenceExample(
   GeofenceManager geofenceManager,
   DeviceManager deviceManager) {
        this.geofenceManager =
   geofenceManager;
        this.deviceManager =
   deviceManager;
    }

    public void createAndAssignGeofence()
    throws Exception {
        // Crear una nueva geocerca
        Geofence geofence = new Geofence
   ();
        geofence.setName("Warehouse Area
   ");
        geofence.setDescription("
   Restricted access area around the
   warehouse");
        geofence.setArea("POLYGON((30 10,
    40 40, 20 40, 10 20, 30 10))"); //
   Ejemplo de  rea  como un pol gono

        // Guardar la geocerca en la base
    de datos
```

```
        geofenceManager.addItem(geofence)
    ;

        // Obtener un dispositivo para
    asociar la geocerca
        Device device = deviceManager.
    getById(1); // Asume que el
    dispositivo con ID 1 existe

        // Asociar la geocerca al
    dispositivo
        geofenceManager.linkDevice(
    geofence.getId(), device.getId());

        System.out.println("Geofence '" +
     geofence.getName() + "' assigned to
    device " + device.getName());
    }
}
```

Explicación:

- Creación de geocerca: Se define una geocerca con un área específica utilizando un polígono.
- Almacenamiento: La geocerca se guarda en la base de datos mediante geofenceManager.
- Asociación a un dispositivo: La geocerca creada se asocia a un dispositivo existente, lo que permite rastrear la entrada y salida de ese dispositivo en la geocerca.

IMPORT 3

```
import org.traccar.model.BaseModel
```

Descripción:
BaseModel es una clase base abstracta en el paquete org.traccar.model que sirve como la superclase para la mayoría de los modelos en Traccar. Proporciona propiedades y métodos comunes a todos los modelos, como identificadores únicos (IDs), marcas de tiempo, y otros atributos que son comunes entre las distintas entidades.
Cómo funciona:
BaseModel define los atributos y métodos que son comunes a todas las entidades que extienden esta clase. Al heredar de BaseModel, las subclases como Device, User, o Geofence obtienen automáticamente estas propiedades y funcionalidades, lo que facilita la gestión y manipulación de objetos en la base de datos.
Funcionalidades principales:

- ID único: Proporciona un campo de identificación único (id) que es común a todos los modelos.
- Timestamps: Incluye campos para gestionar la creación y la última modificación del objeto (created y updated).
- Comparación y hash: Implementa métodos como equals y hashCode para comparar objetos y calcular sus códigos hash de manera consistente.

Dependencias:

- ORM (Object-Relational Mapping): BaseModel depende de un ORM para mapear las propiedades del modelo a columnas de la base de datos.
- Subclases: Es extendido por múltiples clases como Device, User, Geofence, etc., que heredan sus propiedades y métodos.

Ejemplo de Código

```
    import org.traccar.model.BaseModel;

public class CustomModel extends
    BaseModel {
    private String customField;

    public String getCustomField() {
        return customField;
    }

    public void setCustomField(String
    customField) {
        this.customField = customField;
    }

    @Override
    public String toString() {
        return "CustomModel [id=" + getId
    () + ", customField=" + customField +
    "]";
    }
}
```

IMPORT 4

```
import org.traccar.model.Device
```

Descripción:
Device es una clase en el paquete org.traccar.model que representa un dispositivo rastreado en el sistema Traccar. Cada instancia de Device corresponde a un dispositivo físico que reporta su ubicación y otros datos a la plataforma Traccar.
Cómo funciona:
La clase Device contiene propiedades y métodos específicos para gestionar la información asociada a un dispositivo, como su nombre, identificadores únicos (como IMEI), estado, y configuraciones específicas. También maneja la relación del dispositivo con otras entidades como User y Geofence.
Funcionalidades principales:

- Identificación del dispositivo: Almacena identificadores como IMEI o nombres únicos.
- Configuraciones: Permite configurar opciones específicas del dispositivo, como protocolos de comunicación y reportes.
- Estado del dispositivo: Gestiona estados como si el dispositivo está activo, inactivo, o fuera de línea.
- Relaciones: Gestiona las relaciones con usuarios y geocercas, permitiendo la asignación de dispositivos a usuarios o zonas específicas.

Dependencias:

- BaseModel: Hereda de BaseModel, lo que le proporciona un ID único y manejo de timestamps.
- Servicios de base de datos: Depende de los servicios de almacenamiento para persistir y recuperar la información del dispositivo.
- Protocolos de comunicación: Interactúa con diversos protocolos para recibir y enviar datos de rastreo.

Ejemplo de Código

```
    import org.traccar.model.Device;

public class DeviceExample {
    public void createDevice() {
        Device device = new Device();
        device.setName("Device 1");
        device.setUniqueId("ABC123456");
        device.setStatus(Device.
   STATUS_ONLINE);

        // Supongamos que tenemos un
   DeviceManager para manejar el
   dispositivo
        DeviceManager deviceManager = new
    DeviceManager();
        deviceManager.addItem(device);

        System.out.println("Device
   created: " + device.getName());
    }
}
```

IMPORT 5

```
    import org.traccar.model.Group;
```

Descripción:
Group es una clase en el paquete org.traccar.model que representa un grupo de dispositivos en el sistema Traccar. Se utiliza para organizar y gestionar dispositivos de manera agrupada, lo que facilita la administración en entornos con muchos dispositivos.

Cómo funciona:
La clase Group permite la creación y gestión de grupos de dispositivos, proporcionando propiedades como el nombre del grupo y una descripción. Los dispositivos pueden ser asignados a uno o varios grupos, y se pueden aplicar configuraciones comunes a todos los dispositivos dentro de un grupo.

Funcionalidades principales:

- Organización de dispositivos: Permite agrupar dispositivos bajo un nombre y descripción comunes.
- Herencia de configuraciones: Los dispositivos dentro de un grupo pueden heredar configuraciones y políticas establecidas para el grupo.
- Gestión de permisos: Facilita la gestión de permisos y accesos en función de los grupos a los que pertenecen los dispositivos.

Dependencias:

- BaseModel: Hereda de BaseModel, proporcionando atributos comunes como ID y timestamps.
- Relaciones con dispositivos: Depende de la relación con la clase Device para gestionar la pertenencia de los dispositivos a los grupos.
- Servicios de base de datos: Necesita acceso a los servicios de almacenamiento para crear, modificar y eliminar grupos.

Ejemplo de Código

```
    import org.traccar.model.Group;

public class GroupExample {
    public void createGroup() {
        Group group = new Group();
        group.setName("Fleet A");
        group.setDescription("Group for
   Fleet A devices");

        // Supongamos que tenemos un
   GroupManager para manejar el grupo
        GroupManager groupManager = new
   GroupManager();
        groupManager.addItem(group);

        System.out.println("Group created
   : " + group.getName());
    }
}
```

IMPORT 6

```
    import org.traccar.model.User;
```

Descripción:
User es una clase en el paquete org.traccar.model que representa a un usuario en el sistema Traccar. Los usuarios tienen acceso a los dispositivos y configuraciones dentro de la plataforma, y pueden tener roles y permisos específicos.

Cómo funciona:
La clase User gestiona toda la información relacionada con los usuarios del sistema, incluyendo sus credenciales, roles, preferencias y asociaciones con dispositivos y grupos. Además, maneja la autenticación y la autorización para controlar el acceso a los recursos del sistema.

Funcionalidades principales:

- Autenticación: Gestiona las credenciales de usuario (como nombre de usuario y contraseña) y maneja el proceso de autenticación.
- Autorización: Controla los permisos de acceso del usuario a los diferentes recursos del sistema, como dispositivos y reportes.
- Preferencias del usuario: Almacena configuraciones personales, como el idioma preferido y las notificaciones.

- Gestión de relaciones: Asocia usuarios con dispositivos, grupos y otras entidades dentro del sistema.

Dependencias:

- BaseModel: Hereda de BaseModel, lo que le proporciona un ID único y manejo de timestamps.
- Servicios de autenticación: Depende de los servicios de autenticación para validar las credenciales del usuario.
- Gestión de permisos: Interactúa con sistemas de control de acceso para gestionar la autorización de usuarios.

Ejemplo de Código

```
    import org.traccar.model.User;

public class UserExample {
    public void createUser() {
        User user = new User();
        user.setName("John Doe");
        user.setEmail("john.doe@example.
    com");
        user.setPassword("securepassword
    ");

        // Supongamos que tenemos un
    UserManager para manejar al usuario
        UserManager userManager = new
    UserManager();
        userManager.addItem(user);

        System.out.println("User created:
     " + user.getName());
    }
}
```

IMPORT 7

```
    import org.traccar.storage.
    StorageException;
```

Descripción:

StorageException es una clase en el paquete org.traccar.storage que maneja excepciones relacionadas con operaciones de almacenamiento en Traccar. Se utiliza para indicar errores que ocurren durante el acceso, manipulación o persistencia de datos en el sistema de almacenamiento.

Cómo funciona:

StorageException extiende de la clase Exception y se lanza cuando ocurre un error durante las operaciones de almacenamiento, como al guardar o recuperar datos desde la base de datos. Al capturar esta excepción, el sistema puede manejar adecuadamente los errores de almacenamiento, ya sea mostrando mensajes de error al usuario o intentando una recuperación.

Funcionalidades principales:

- Manejo de errores de almacenamiento: Proporciona una forma de capturar y manejar errores específicos del almacenamiento de datos.
- Detección de problemas: Ayuda a identificar problemas específicos como la conexión fallida a la base de datos, violaciones de integridad, o errores en las consultas.
- Logging y monitoreo: Permite el registro detallado de excepciones, lo que facilita el monitoreo y la depuración de problemas en el sistema.

Dependencias:

- Operaciones de base de datos: Depende de los servicios de base de datos y el ORM para gestionar las excepciones durante las operaciones de almacenamiento.
- Sistemas de logging: Puede integrarse con sistemas de logging para registrar las excepciones y sus detalles.

Ejemplo de Código

```
    import org.traccar.storage.
    StorageException;

public class StorageExceptionExample {
    public void handleException() {
        try {
            // Simulaci n de una
    operaci n que puede lanzar
    StorageException
            throw new StorageException("
    Error accessing storage");
        } catch (StorageException e) {
            System.err.println("Storage
    error: " + e.getMessage());
        }
    }
}
```

IMPORT 8

```
    import org.traccar.storage.query.
    Columns;
```

Descripción:

Columns es una clase en el paquete org.traccar.storage.query que se utiliza para especificar las columnas de la base de datos que se desean seleccionar en una consulta. Es parte del mecanismo de consultas dinámicas de Traccar.

Cómo funciona:

La clase Columns permite definir de manera programática las columnas que se deben incluir en el resultado de una consulta SQL. Esto es útil para optimizar consultas y limitar la cantidad de datos recuperados, seleccionando solo las columnas necesarias.

Funcionalidades principales:

- Selección de columnas: Permite definir explícitamente qué columnas de la tabla se deben incluir en los resultados de la consulta.
- Optimización de consultas: Al seleccionar solo las columnas necesarias, mejora el rendimiento de las consultas al reducir la cantidad de datos transferidos.

- Consultas dinámicas: Facilita la construcción de consultas SQL dinámicas y personalizadas en función de las necesidades de la aplicación.

Dependencias:

- Request: Generalmente se utiliza junto con la clase Request para construir consultas completas.
- ORM: Depende del ORM para mapear las columnas seleccionadas a los atributos de los modelos.

Ejemplo de Código

```
    import org.traccar.storage.query.
   Columns;
import org.traccar.storage.query.Request;
import org.traccar.model.Device;

public class ColumnsExample {
    public void queryWithColumns() {
        // Seleccionar columnas
   espec ficas
        Columns columns = new Columns("
   name", "uniqueId");

        // Crear una solicitud de
   consulta con las columnas
   seleccionadas
        Request request = new Request(
   columns, null);

        // Supongamos que tenemos un
   DeviceManager para manejar la consulta
        DeviceManager deviceManager = new
    DeviceManager();
        List<Device> devices =
   deviceManager.query(request);

        for (Device device : devices) {
            System.out.println("Device
   Name: " + device.getName() + ", Unique
    ID: " + device.getUniqueId());
        }
    }
}
```

IMPORT 9

```
    import org.traccar.storage.query.
   Condition;
```

Descripción:
Condition es una clase en el paquete org.traccar.storage.query que se utiliza para definir condiciones en una consulta SQL. Estas condiciones determinan los criterios bajo los cuales se seleccionan, actualizan o eliminan registros en la base de datos.

Cómo funciona:
La clase Condition permite crear y combinar condiciones para consultas SQL de manera programática. Estas condiciones se traducen en cláusulas WHERE, JOIN, etc., que filtran los registros según los criterios especificados.

Funcionalidades principales:

- Definición de filtros: Permite crear condiciones como igualdad, desigualdad, mayor que, menor que, entre otras, para filtrar los resultados de la consulta.
- Combinación de condiciones: Facilita la combinación de múltiples condiciones utilizando operadores lógicos como AND y OR.
- Consultas dinámicas: Ayuda a construir consultas SQL flexibles y dinámicas basadas en las condiciones especificadas.

Dependencias:

- Request: Generalmente se utiliza junto con la clase Request para construir consultas completas con filtros y condiciones.
- ORM: Depende del ORM para mapear las condiciones a las consultas SQL subyacentes.

Ejemplo de Código

```
    import org.traccar.storage.query.
   Condition;
import org.traccar.storage.query.Request;
import org.traccar.model.Device;

public class ConditionExample {
    public void queryWithCondition() {
        // Crear una condici n para
   filtrar dispositivos por nombre
        Condition condition = new
   Condition.Equals("name", "Device 1");

        // Crear una solicitud de
   consulta con la condici n
        Request request = new Request(new
    Columns.All(), condition);

        // Supongamos que tenemos un
   DeviceManager para manejar la consulta
        DeviceManager deviceManager = new
    DeviceManager();
        List<Device> devices =
   deviceManager.query(request);

        for (Device device : devices) {
            System.out.println("Filtered
   Device: " + device.getName());
        }
    }
}
```

IMPORT 10

```
    import org.traccar.storage.query.
   Request;
```

Descripción:
Request es una clase en el paquete org.traccar.storage.query que representa una consulta completa en el sistema Traccar. Esta clase encapsula la selección de columnas, las condiciones, el orden y otros parámetros necesarios para ejecutar una consulta SQL.

Cómo funciona:
Request permite construir consultas SQL de manera programática, incluyendo qué columnas seleccionar, qué condiciones aplicar, cómo ordenar los resultados, y otros aspectos de la consulta. Una vez construida, la consulta se puede ejecutar para obtener los resultados deseados.

Funcionalidades principales:

- Construcción de consultas: Facilita la creación de consultas SQL completas con todas las partes necesarias, como selección de columnas, condiciones, y orden.
- Ejecución de consultas: Una vez construida, la consulta se puede ejecutar para obtener resultados desde la base de datos.
- Consultas dinámicas: Soporta la construcción de consultas dinámicas y flexibles, que pueden adaptarse a diferentes necesidades de la aplicación.

Dependencias:

- Columns y Condition: Depende de estas clases para definir qué columnas seleccionar y qué condiciones aplicar en la consulta.
- ORM: Depende del ORM para mapear la consulta a la base de datos y obtener los resultados.

Ejemplo de Código

```
    import org.traccar.storage.query.
   Columns;
import org.traccar.storage.query.
   Condition;
import org.traccar.storage.query.Request;
import org.traccar.model.Device;

public class RequestExample {
    public void performQuery() {
        // Definir columnas a seleccionar
    y condici n de filtro
        Columns columns = new Columns("
   name", "status");
        Condition condition = new
   Condition.Equals("status", Device.
   STATUS_ONLINE);

        // Crear una solicitud de
   consulta
        Request request = new Request(
   columns, condition);

        // Supongamos que tenemos un
   DeviceManager para manejar la consulta
        DeviceManager deviceManager = new
    DeviceManager();
        List<Device> devices =
   deviceManager.query(request);

        for (Device device : devices) {
            System.out.println("Device
   Name: " + device.getName() + ", Status
   : " + device.getStatus());
        }
    }
}
```

IMPORT 11

```
    import org.traccar.geofence.
   GeofenceCircle;
```

Descripción:
GeofenceCircle es una clase en el paquete org.traccar.geofence que representa una geocerca circular en el sistema Traccar. Esta clase define un área circular basada en un punto central (latitud y longitud) y un radio, que se puede utilizar para monitorear si un dispositivo entra o sale de este área.

Cómo funciona:
GeofenceCircle permite crear y gestionar geocercas circulares que pueden ser utilizadas para rastrear la ubicación de dispositivos en tiempo real. Al definir una geocerca, es posible generar alertas cuando un dispositivo cruza los límites de la misma, ya sea entrando o saliendo del área definida.

Funcionalidades principales:

- Definición de geocercas: Facilita la creación de geocercas basadas en un punto central y un radio.
- Monitoreo de dispositivos: Permite rastrear la entrada y salida de dispositivos dentro de la geocerca definida.
- Flexibilidad en configuración: Soporta la definición de geocercas con diferentes radios y ubicaciones, adaptándose a diversas necesidades de monitoreo.

Dependencias:

- Latitud y Longitud: Depende de estos parámetros para definir el punto central de la geocerca.
- Sistema de alertas: Integra con el sistema de alertas para notificar eventos relacionados con la geocerca.

Ejemplo de Código

```
    import org.traccar.geofence.
   GeofenceCircle;
import org.traccar.model.Position;

public class GeofenceExample {
    public void defineGeofence() {
        // Definir el centro de la
   geocerca y su radio
        double latitude = 40.7128;
        double longitude = -74.0060;
        double radius = 500; // Radio en
   metros
```

```
        // Crear una geocerca circular
        GeofenceCircle geofence = new
    GeofenceCircle(latitude, longitude,
    radius);

        // Supongamos que tenemos un
    PositionManager para manejar la
    ubicaci n del dispositivo
        PositionManager positionManager =
     new PositionManager();
        Position currentPosition =
    positionManager.getCurrentPosition();

        // Verificar si el dispositivo
    est  dentro de la geocerca
        if (geofence.contains(
    currentPosition.getLatitude(),
    currentPosition.getLongitude())) {
            System.out.println("El
    dispositivo est  dentro de la
    geocerca.");
        } else {
            System.out.println("El
    dispositivo est  fuera de la geocerca
    .");
        }
    }
}
```

IMPORT 12

```
    import org.traccar.geofence.
    GeofenceGeometry;
```

Descripción:
GeofenceGeometry es una clase en el paquete org.traccar.geofence que representa la geometría de una geocerca en el sistema Traccar. Esta clase define las formas y límites geográficos de una geocerca, lo que puede incluir círculos, polígonos, y otras formas geométricas complejas que se utilizan para delimitar áreas en las que se monitorean dispositivos.

Cómo funciona:
GeofenceGeometry proporciona una abstracción que permite definir geocercas de diferentes formas y tamaños. La clase maneja la lógica para determinar si un dispositivo se encuentra dentro o fuera de los límites definidos por la geocerca, lo que facilita la configuración de zonas de monitoreo complejas en aplicaciones de rastreo.

Funcionalidades principales:

- Definición de geocercas complejas: Permite crear geocercas no solo circulares, sino también polígonos y otras formas geométricas.
- Monitoreo preciso: Facilita el monitoreo detallado de dispositivos dentro de zonas con formas geométricas complejas.
- Flexibilidad en la configuración: Soporta diversas configuraciones de geocercas para adaptarse a diferentes requisitos de monitoreo.

Dependencias:

- Puntos geográficos: Depende de coordenadas (latitud y longitud) para definir la geometría de la geocerca.
- Sistema de alertas: Se integra con el sistema de alertas para notificar eventos cuando un dispositivo entra o sale de la geocerca.

Ejemplo de Código

```
    import org.traccar.geofence.
    GeofenceGeometry;
import org.traccar.model.Position;
import java.util.Arrays;

public class GeofenceGeometryExample {
    public void defineGeofence() {
        // Definir los puntos que
    conforman la geometr a de la geocerca
     (en este caso, un tri ngulo)
        double[][] points = {
            {40.7128, -74.0060},
            {40.7308, -73.9975},
            {40.7188, -73.9857}
        };

        // Crear una geocerca geom trica
     basada en los puntos
        GeofenceGeometry geofence = new
    GeofenceGeometry(Arrays.asList(points)
    );

        // Supongamos que tenemos un
    PositionManager para manejar la
    ubicaci n del dispositivo
        PositionManager positionManager =
     new PositionManager();
        Position currentPosition =
    positionManager.getCurrentPosition();

        // Verificar si el dispositivo
    est  dentro de la geocerca
    geom trica
        if (geofence.contains(
    currentPosition.getLatitude(),
    currentPosition.getLongitude())) {
            System.out.println("El
    dispositivo est  dentro de la
    geocerca geom trica.");
        } else {
            System.out.println("El
    dispositivo est  fuera de la geocerca
     geom trica.");
        }
    }
```

```
}
```

IMPORT 13

```
import org.traccar.geofence.
GeofencePolygon;
```

Descripción:
GeofencePolygon es una clase en el paquete org.traccar.geofence que representa una geocerca poligonal en el sistema Traccar. Esta clase define un área geográfica en forma de polígono, que se puede utilizar para delimitar zonas específicas y monitorear si los dispositivos entran o salen de estas áreas.

Cómo funciona:
GeofencePolygon permite crear geocercas basadas en múltiples puntos geográficos conectados, formando un polígono cerrado. Esta clase es útil para definir áreas de monitoreo con formas más complejas que un círculo, como zonas urbanas, parques, o propiedades con bordes irregulares.

Funcionalidades principales:

- Definición de geocercas poligonales: Facilita la creación de geocercas con múltiples vértices que forman un área cerrada.
- Monitoreo de zonas complejas: Permite el rastreo de dispositivos dentro de áreas con formas geométricas más detalladas.
- Flexibilidad en configuración: Soporta la definición de polígonos de cualquier forma, adaptándose a diversos requerimientos de monitoreo.

Dependencias:

- Coordenadas geográficas: Depende de un conjunto de coordenadas (latitud y longitud) para definir los vértices del polígono.
- Sistema de alertas: Se integra con el sistema de alertas para notificar eventos de entrada y salida del área poligonal.

Ejemplo de Código

```
import org.traccar.geofence.
GeofencePolygon;
import org.traccar.model.Position;
import java.util.Arrays;

public class GeofencePolygonExample {
    public void definePolygonGeofence() {
        // Definir los puntos que
    conforman la geocerca poligonal
        double[][] points = {
            {40.7128, -74.0060},
            {40.7308, -73.9975},
            {40.7188, -73.9857},
            {40.7088, -74.0010}
        };

        // Crear una geocerca poligonal
    basada en los puntos
        GeofencePolygon geofence = new
    GeofencePolygon(Arrays.asList(points))
    ;

        // Supongamos que tenemos un
    PositionManager para manejar la
    ubicaci n del dispositivo
        PositionManager positionManager =
     new PositionManager();
        Position currentPosition =
    positionManager.getCurrentPosition();

        // Verificar si el dispositivo
    est  dentro de la geocerca poligonal
        if (geofence.contains(
    currentPosition.getLatitude(),
    currentPosition.getLongitude())) {
            System.out.println("El
    dispositivo est  dentro de la
    geocerca poligonal.");
        } else {
            System.out.println("El
    dispositivo est  fuera de la geocerca
     poligonal.");
        }
    }
}
```

IMPORT 14

```
import org.traccar.geofence.
GeofencePolyline;
```

Descripción:
GeofencePolyline es una clase en el paquete org.traccar.geofence que representa una geocerca en forma de polilínea en el sistema Traccar. A diferencia de los polígonos, una polilínea no es un área cerrada, sino una serie de líneas conectadas que pueden utilizarse para definir rutas o caminos.

Cómo funciona:
GeofencePolyline permite la creación de geocercas lineales que siguen un conjunto de puntos geográficos, formando una ruta o camino. Esta clase es ideal para monitorear si un dispositivo sigue o se desvía de una ruta predefinida.

Funcionalidades principales:

- Definición de rutas: Facilita la creación de geocercas basadas en caminos o rutas, utilizando polilíneas.
- Monitoreo de desplazamientos: Permite rastrear si un dispositivo sigue la ruta definida o se desvía de ella.
- Flexibilidad en configuración: Soporta la definición de polilíneas de cualquier forma, adaptándose a diferentes necesidades de seguimiento.

Dependencias:

- Coordenadas geográficas: Depende de un conjunto de puntos (latitud y longitud) para definir la trayectoria de la polilínea.
- Sistema de alertas: Se integra con el sistema de alertas para notificar desviaciones o eventos relacionados con la polilínea.

Ejemplo de Código

```
    import org.traccar.geofence.
   GeofencePolyline;
import org.traccar.model.Position;
import java.util.Arrays;

public class GeofencePolylineExample {
    public void definePolylineGeofence()
   {
        // Definir los puntos que
   conforman la geocerca polil nea
        double[][] points = {
            {40.7128, -74.0060},
            {40.7308, -73.9975},
            {40.7188, -73.9857}
        };

        // Crear una geocerca polil nea
   basada en los puntos
        GeofencePolyline geofence = new
   GeofencePolyline(Arrays.asList(points)
   );

        // Supongamos que tenemos un
   PositionManager para manejar la
   ubicaci n del dispositivo
        PositionManager positionManager =
    new PositionManager();
        Position currentPosition =
   positionManager.getCurrentPosition();

        // Verificar si el dispositivo
   sigue la polil nea
        if (geofence.isNearPolyline(
   currentPosition.getLatitude(),
   currentPosition.getLongitude())) {
            System.out.println("El
   dispositivo sigue la ruta de la
   polil nea.");
        } else {
            System.out.println("El
   dispositivo se ha desviado de la ruta
   .");
        }
    }
}
```

IMPORT 15

```
    import org.traccar.storage.
   QueryIgnore;
```

Descripción:
QueryIgnore es una anotación en el paquete org.traccar.storage que se utiliza para indicar que un campo o método debe ser ignorado durante la construcción de consultas SQL en el sistema Traccar. Esta anotación es útil cuando ciertos atributos de una clase no deben ser considerados en operaciones de almacenamiento o consulta en la base de datos.

Cómo funciona:
QueryIgnore se aplica sobre los campos o métodos de una clase que no deben ser incluidos en las operaciones de consulta o actualización en la base de datos. Esto es útil para omitir información que es irrelevante para la persistencia o que debe ser gestionada de manera independiente.

Funcionalidades principales:

- Exclusión de campos: Facilita la omisión de ciertos campos o métodos durante la construcción de consultas SQL.
- Control sobre persistencia: Permite un mayor control sobre qué datos se almacenan o consultan desde la base de datos.
- Integración sencilla: Se integra fácilmente con otras anotaciones y configuraciones en el sistema de persistencia.

Dependencias:

- Sistema de ORM: Depende del ORM para aplicar la exclusión de campos durante las operaciones de consulta y almacenamiento.
- Configuración de clases: Requiere que las clases estén correctamente configuradas para utilizar esta anotación de manera efectiva.

Ejemplo de Código

```
    import org.traccar.storage.
   QueryIgnore;

public class Device {

    private String name;

    @QueryIgnore
    private String temporaryStatus;

    // Otros campos y m todos...

    public String getName() {
        return name;
    }

    public String getTemporaryStatus() {
        return temporaryStatus;
    }

    public void setTemporaryStatus(String
     temporaryStatus) {
```

```
        this.temporaryStatus =
    temporaryStatus;
     }
}
```

IMPORT 16

```
    import org.traccar.storage.
    StorageName;
```

Descripción:
StorageName es una anotación en el paquete org.traccar.storage que se utiliza para especificar el nombre de la tabla en la base de datos que corresponde a una clase en el sistema Traccar. Esta anotación es esencial para mapear correctamente las clases de modelo a las tablas correspondientes en la base de datos relacional.

Cómo funciona:
StorageName se aplica a nivel de clase y define explícitamente el nombre de la tabla en la base de datos que debe utilizarse para almacenar instancias de esa clase. Esto es particularmente útil cuando el nombre de la tabla no sigue las convenciones de nomenclatura por defecto o cuando se requiere un nombre específico por razones de compatibilidad o legado.

Funcionalidades principales:

- Mapeo de tablas: Facilita la asociación directa entre una clase de modelo y una tabla en la base de datos.
- Flexibilidad en nombres: Permite el uso de nombres de tablas personalizados, mejorando la flexibilidad en la estructura de la base de datos.
- Integración con ORM: Se integra con el sistema ORM para garantizar que las operaciones de persistencia utilicen el nombre de tabla correcto.

Dependencias:

- Sistema de ORM: Depende del ORM para utilizar el nombre de la tabla especificado durante las operaciones de consulta y almacenamiento.
- Configuración de clases: Requiere que las clases de modelo estén configuradas correctamente para utilizar esta anotación.

Ejemplo de Código

```
    import org.traccar.storage.
    StorageName;

@StorageName("device_table")
public class Device {

    private String name;
    private String status;

    // Otros campos y m todos...

    public String getName() {
        return name;
    }

    public String getStatus() {
        return status;
    }
}
```

IMPORT 17

```
    import org.traccar.config.Config;
```

Descripción:
Config es una clase en el paquete org.traccar.config que maneja la configuración global del sistema Traccar. Esta clase permite cargar, acceder y gestionar las diferentes configuraciones que son esenciales para el funcionamiento del sistema, tales como parámetros de red, opciones de almacenamiento, y ajustes de servidor.

Cómo funciona:
Config carga la configuración desde archivos, variables de entorno, o valores predeterminados, y proporciona métodos para acceder a estos parámetros desde cualquier parte del código. Esto asegura que las configuraciones sean centralizadas y fáciles de gestionar, facilitando la administración y modificación del comportamiento del sistema.

Funcionalidades principales:

- Carga de configuración: Facilita la carga de parámetros desde archivos de configuración o variables de entorno.
- Acceso centralizado: Proporciona una interfaz unificada para acceder a los valores de configuración en todo el sistema.
- Soporte para múltiples formatos: Soporta diferentes formatos de configuración, como propiedades, YAML, y más.

Dependencias:

- Sistema de archivos: Depende del sistema de archivos para cargar configuraciones desde archivos locales.
- Variables de entorno: Puede depender de variables de entorno para obtener configuraciones dinámicas.

Ejemplo de Código

```
    import org.traccar.config.Config;

public class ServerConfiguration {

    public void loadConfiguration() {
        // Crear una instancia de Config
    y cargar configuraciones
        Config config = new Config();
        config.load("config.properties");

        // Acceder a un par metro de
    configuraci n
        String databaseUrl = config.
    getString("database.url");
        System.out.println("Database URL:
     " + databaseUrl);
```

```
    }
}
```

IMPORT 18

```
    import org.traccar.helper.
   DistanceCalculator;
```

Descripción:
DistanceCalculator es una clase en el paquete org.traccar.helper que proporciona métodos para calcular distancias entre puntos geográficos en el sistema Traccar. Esta clase es fundamental para aplicaciones que requieren medir la distancia entre dispositivos, geocercas u otros puntos geográficos.

Cómo funciona:
DistanceCalculator utiliza fórmulas matemáticas, como la fórmula de Haversine, para calcular la distancia entre dos puntos dados por sus coordenadas de latitud y longitud. Esto es útil en una variedad de escenarios, como calcular la distancia recorrida por un dispositivo o verificar si un dispositivo se encuentra dentro de un radio específico de una geocerca.

Funcionalidades principales:

- Cálculo de distancias: Permite calcular distancias precisas entre dos puntos geográficos.
- Soporte para diferentes unidades: Proporciona métodos para calcular distancias en diferentes unidades, como metros y kilómetros.
- Aplicaciones diversas: Se puede utilizar en varios escenarios, incluyendo monitoreo de geocercas y seguimiento de rutas.

Dependencias:

- Coordenadas geográficas: Depende de coordenadas (latitud y longitud) para realizar los cálculos de distancia.
- Algoritmos matemáticos: Utiliza algoritmos matemáticos específicos para el cálculo preciso de distancias.

Ejemplo de Código

```
    import org.traccar.helper.
   DistanceCalculator;

public class DistanceExample {

    public void calculateDistance() {
        // Definir dos puntos
   geogr ficos
        double lat1 = 40.7128;
        double lon1 = -74.0060;
        double lat2 = 34.0522;
        double lon2 = -118.2437;

        // Calcular la distancia entre
   los dos puntos
        double distance =
   DistanceCalculator.distance(lat1, lon1
   , lat2, lon2);

        System.out.println("Distance
   between points: " + distance + "
   kilometers.");
    }
}
```

IMPORT 19

```
    import org.traccar.model.Geofence;
```

Descripción:
Geofence es una clase en el paquete org.traccar.model que representa una geocerca en el sistema Traccar. Esta clase encapsula la definición, atributos, y comportamiento de una geocerca, permitiendo la creación, modificación y monitoreo de áreas geográficas específicas donde se desea rastrear la actividad de dispositivos.

Cómo funciona:
Geofence permite definir áreas geográficas utilizando diferentes formas geométricas, como círculos, polígonos, y más. Una vez definida, la geocerca puede utilizarse para monitorear la entrada o salida de dispositivos de esa área, generando eventos y alertas en función del comportamiento configurado.

Funcionalidades principales:

- Definición de geocercas: Facilita la creación de geocercas con diferentes formas y tamaños.
- Monitoreo de dispositivos: Permite rastrear la actividad de dispositivos dentro y fuera de la geocerca.
- Generación de eventos: Soporta la creación de eventos y alertas basados en la interacción de dispositivos con la geocerca.

Dependencias:

- Coordenadas geográficas: Depende de coordenadas (latitud y longitud) para definir los límites de la geocerca.
- Sistema de eventos: Se integra con el sistema de eventos para notificar cuando un dispositivo entra o sale de la geocerca.

Ejemplo de Código

```
    import org.traccar.model.Geofence;
import org.traccar.model.Position;

public class GeofenceExample {

    public void createGeofence()


\section*{IMPORT 20}
\begin{lstlisting}
    import org.traccar.model.Device
```

IMPORT 21

```
    import org.traccar.config.Keys;
```

Descripción:
Keys es una clase en el paquete org.traccar.config que define las claves de configuración utilizadas en el sistema Traccar. Estas claves representan los nombres de los parámetros de configuración que pueden ser utilizados para ajustar el comportamiento del sistema, como la configuración del servidor, la base de datos, las opciones de rastreo, entre otros.

Cómo funciona:
Keys actúa como un repositorio centralizado de todas las claves de configuración, proporcionando una manera consistente y segura de acceder a los parámetros de configuración. Estas claves son utilizadas junto con la clase 'Config' para recuperar y establecer valores de configuración dentro de la aplicación Traccar.

Funcionalidades principales:

- Definición centralizada de claves: Proporciona una ubicación única para todas las claves de configuración utilizadas en el sistema.
- Seguridad en la configuración: Facilita la gestión de las configuraciones evitando errores al usar claves definidas de manera inconsistente.
- Integración con Config: Se utiliza en combinación con la clase 'Config' para recuperar valores de configuración específicos.

Dependencias:

- Config: Depende de la clase 'Config' para utilizar las claves definidas y acceder a los valores de configuración.
- Sistema de archivos: Los valores de las claves pueden ser cargados desde archivos de configuración o variables de entorno.

Ejemplo de Código

```
    import org.traccar.config.Config;
import org.traccar.config.Keys;

public class ServerConfigExample {

    public void loadServerPort() {
        // Crear una instancia de Config
    y cargar configuraciones
        Config config = new Config();
        config.load("config.properties");

        // Utilizar una clave definida en
     Keys para acceder a un par metro de
    configuraci n
        int serverPort = config.
    getInteger(Keys.SERVER_PORT);
        System.out.println("Server Port:
    " + serverPort);
    }
}
```

IMPORT 22

```
    import org.traccar.config.Keys;
```

Descripción:
Hashing es una clase en el paquete org.traccar.helper que proporciona métodos para generar valores hash a partir de datos de entrada. Esta clase es utilizada para realizar operaciones de hashing como parte de procesos de seguridad, validación o comparación de datos.

Cómo funciona:
Hashing proporciona métodos estáticos que toman datos de entrada y generan un valor hash, que es una representación única del conjunto de datos. Esto es útil para verificar la integridad de los datos o para almacenar contraseñas de manera segura.

Funcionalidades principales:

- Generación de hash: Facilita la creación de valores hash a partir de datos de entrada.
- Seguridad: Utiliza algoritmos de hashing seguros para proteger contraseñas y datos sensibles.
- Comparación de datos: Permite comparar datos mediante sus valores hash en lugar de los datos originales.

Dependencias:

- Algoritmos de hashing: Depende de los algoritmos de hashing proporcionados para generar valores hash.
- Biblioteca de seguridad: Puede requerir bibliotecas adicionales para operaciones criptográficas.

Ejemplo de Código

```
public class User {
private String password;

public void setPassword(String password)
    {
    this.password = Hashing.hash(password
    );
}

// Otros campos y m todos...

public String getPassword() {
    return password;
}
}
```

V. Resultados

- Se logroaron detallar exitosamente los conceptos y métodos GET, POST, PUT y DELETE de la API Geofences de Traccar, permitiendo la gestión completa de geocercas dentro del sistema
- La documentación de la API de Geofences fue realizada con éxito, detallando cada uno de los métodos CRUD (Crear, Leer, Actualizar, Eliminar) y proporcionando instrucciones claras para su implementación en un entorno Dockerizado. Se lograron ejecutar y validar correctamente los comandos correspondientes a los métodos GET, POST, PUT y DELETE, comprobando que cada

operación se reflejara de manera adecuada tanto en la API como en la interfaz web, lo que demuestra la efectividad y funcionalidad de la API en la gestión de geocercas.

VI. Discusión y Conclusiones

- Esta API es fundamental para la gestión eficiente de geocercas, permitiendo realizar operaciones clave como la creación, actualización, eliminación y consulta de geocercas mediante los métodos GET, POST, PUT y DELETE. La implementación de estos métodos en un entorno dockerizado no solo facilita la escalabilidad y consistencia en diferentes entornos, sino que también asegura una gestión más controlada y eficiente de las geocercas en sistemas de rastreo en tiempo real. Esto refuerza la utilidad de Traccar como una plataforma robusta para la gestión de dispositivos y áreas geográficas monitoreadas.
- La guía para la documentación e implementación de la API de Geofences en un entorno Dockerizado evidencia la capacidad de Docker para manejar de manera eficiente la API, permitiendo una gestión centralizada y coherente de las geocercas. La utilización de herramientas como PowerShell y Bash en conjunto con Docker asegura que las operaciones sobre las geocercas se realicen de forma controlada y reproducible, optimizando el proceso de desarrollo y despliegue en entornos de producción.

Referencias

[1] C. Pereira Villalba, J. M. Oviedo Salinas, and D. I. Kang Cardozo, "Rastreo de transporte público interurbano de ciudad del este aplicando plataformas de código abierto," 2017.

[2] W. C. Ordoñez, C. A. Poveda, and D. Rodríguez, "Sistema de seguimiento gps para la optimización de rutas de distribución en última milla," *Mare Ingenii*, vol. 2, no. 2, pp. 16–40, 2020.

[3] "API Reference - TracCar." [Online]. Available: https://www.traccar.org/api-reference/#tag/Geofences/paths/~1geofences~1%7Bid%7D/put