API StatisticsResource

***Resumen*—En este documento se presenta la documentación de la API `StatisticsResource` del sistema Traccar, la cual proporciona un servicio RESTful para la obtención de estadísticas de la plataforma en un rango de fechas específico. Se sigue un enfoque estructurado para documentar la API, cubriendo aspectos esenciales como el título, descripción general, método HTTP, parámetros de entrada, y ejemplos de solicitud y respuesta. La documentación clara y detallada de esta API asegura que los desarrolladores puedan integrarla eficientemente en sus aplicaciones, mejorando la capacidad de los administradores del sistema para acceder y analizar datos de rendimiento.**

***Abstract*—This document presents the documentation of the `StatisticsResource` API from the Traccar system, which provides a RESTful service for obtaining platform statistics within a specific date range. A structured approach is followed to document the API, covering essential aspects such as title, general description, HTTP method, input parameters, and request-response examples. The clear and detailed documentation of this API ensures that developers can efficiently integrate it into their applications, enhancing the system administrators' ability to access and analyze performance data.**

***Index Terms*—API documentation, RESTful service, Traccar, StatisticsResource, system performance, data analysis**

I. Introducción

En esta documentación, se presenta la API `StatisticsResource` del sistema Traccar, la cual permite a los administradores obtener estadísticas sobre la plataforma en un rango de fechas determinado. La API está diseñada para ser utilizada por usuarios con permisos de administrador, garantizando que solo personal autorizado pueda acceder a estos datos sensibles.

Traccar es una plataforma de rastreo GPS de código abierto que ofrece servicios de seguimiento en tiempo real de vehículos, personas, y otros activos [1]. Desde su lanzamiento, ha ganado popularidad debido a su flexibilidad y escalabilidad, permitiendo a los usuarios implementar soluciones de rastreo personalizadas en una amplia variedad de contextos, desde pequeñas flotas hasta grandes operaciones logísticas [2].

La API de estadísticas es una herramienta esencial dentro de esta plataforma, ya que permite a los administradores analizar ymonitorear el rendimiento y uso del sistema a través de las métricas capturadas. Las estadísticas obtenidas pueden incluir datos sobre la frecuencia de reportes, la precisión del posicionamiento y el tiempo de inactividad del sistema, todos ellos cruciales para la toma de decisiones basada en datos [3]. Además, la API está diseñada para integrarse fácilmente con otras herramientas de análisis de datos, como sistemas de Business Intelligence (BI), lo que amplía sus capacidades y permite un análisis más profundo [4].

Documentar correctamente una API es fundamental para asegurar que los desarrolladores puedan utilizarla de manera efectiva, facilitando su integración en sistemas existentes y garantizando la correcta interpretación de las respuestas generadas [5]. En los siguientes apartados, se detalla la estructura de la API, incluyendo los métodos disponibles, los parámetros de entrada, la estructura de las respuestas y ejemplos prácticos de uso. Esta documentación tiene como objetivo proporcionar una guía completa para desarrolladores que deseen integrar o utilizar esta API en sus aplicaciones o sistemas.

Identify applicable funding agency here. If none, delete this.

II. Materiales y Métodos

A. Materiales

- Computador Windows
- Procesador i5 13va Gen
- Tarjeta Gráfica NVIDIA RTX 3050 6gb
- 1tb SSD

B. Metodos

- Documentación API
- Experimentación
- Revisión bibliográfica

III. Desarrollo

En esta sección, se documenta el código fuente de la clase `StatisticsResource` que forma parte del sistema Traccar. Esta clase es responsable de proporcionar la funcionalidad para obtener estadísticas del sistema en un rango de fechas específico. A continuación, se presenta el código detallado y su respectiva explicación.

A. Código Fuente

```
/*
 * Copyright 2016 - 2022 Anton Tananaev (
     anton@traccar.org)
 *
 * Licensed under the Apache License, Version 2.0 (
     the "License");
 * you may not use this file except in compliance
     with the License.
 * You may obtain a copy of the License at
 *
 *    http://www.apache.org/licenses/LICENSE-2.0
 *
```

```
 * Unless required by applicable law or agreed to in
     writing, software
 * distributed under the License is distributed on
    an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND,
    either express or implied.
 * See the License for the specific language
    governing permissions and
 * limitations under the License.
 */
package org.traccar.api.resource;

import org.traccar.api.BaseResource;
import org.traccar.model.Statistics;
import org.traccar.storage.StorageException;
import org.traccar.storage.query.Columns;
import org.traccar.storage.query.Condition;
import org.traccar.storage.query.Order;
import org.traccar.storage.query.Request;

import jakarta.ws.rs.Consumes;
import jakarta.ws.rs.GET;
import jakarta.ws.rs.Path;
import jakarta.ws.rs.Produces;
import jakarta.ws.rs.QueryParam;
import jakarta.ws.rs.core.MediaType;
import java.util.Collection;
import java.util.Date;

@Path("statistics")
@Produces(MediaType.APPLICATION_JSON)
@Consumes(MediaType.APPLICATION_JSON)
public class StatisticsResource extends BaseResource
     {

   @GET
   public Collection<Statistics> get(
         @QueryParam("from") Date from, @QueryParam(
             "to") Date to) throws StorageException
             {
      permissionsService.checkAdmin(getUserId());
      return storage.getObjects(Statistics.class,
          new Request(
            new Columns.All(),
            new Condition.Between("captureTime", "
                from", from, "to", to),
            new Order("captureTime")));
   }

}
```

B. Descripción del Código

- **Licencia:** El bloque de comentarios al inicio del código especifica que este se encuentra bajo la licencia Apache 2.0, indicando que puede ser usado y modificado bajo los términos de esta licencia.
- **Paquete:** La clase `StatisticsResource` se encuentra dentro del paquete `org.traccar.api.resource`, el cual contiene las clases relacionadas con la API del sistema.
- **Importaciones:** Se importan varias clases necesarias para la funcionalidad, como `BaseResource`, `Statistics`, y clases relacionadas con el manejo de almacenamiento, consultas y peticiones en la base de datos.
- **Anotaciones:**
  - `@Path("statistics")`: Define la ruta en la cual esta API estará disponible.
  - `@Produces(MediaType.APPLICATION_JSON)`: Especifica que la respuesta de la API será en formato JSON.
  - `@Consumes(MediaType.APPLICATION_JSON)`: Indica que la API espera solicitudes en formato JSON.
- **Método `get`:** Este es el método principal de la clase, que se encarga de manejar las solicitudes GET a la API.
  - `@GET`: Indica que el método responde a solicitudes HTTP GET.
  - `public Collection<Statistics> get(@QueryParam("from") Date from, @QueryParam("to") Date to) throws StorageException`: Define que el método toma dos parámetros de consulta, `from` y `to`, ambos de tipo `Date`, y devuelve una colección de objetos `Statistics`.
  - `permissionsService.checkAdmin(getUserId())`: Verifica que el usuario actual tenga permisos de administrador antes de procesar la solicitud.
  - `return storage.getObjects(Statistics.class, new Request(...))`: Realiza la consulta a la base de datos para obtener las estadísticas dentro del rango de fechas especificado, ordenadas por la hora de captura (`captureTime`).

C. Documentación de las Importaciones

A continuación, se detalla el propósito de cada una de las importaciones utilizadas en la clase `StatisticsResource` y su relevancia para el correcto funcionamiento de la API.

- **`import org.traccar.api.BaseResource;`**
  - **Descripción:** Esta importación hace referencia a la clase `BaseResource`, que es una clase base para todos los recursos de la API en el sistema Traccar. Proporciona métodos y funcionalidades comunes que son compartidas por todas las API, como el manejo de autenticación y permisos. La clase `BaseResource` facilita la interacción con los servicios de seguridad y almacenamiento, que son esenciales para el funcionamiento seguro y eficiente de las APIs en el sistema Traccar.
  - **Función en la API:** La clase `BaseResource` es crucial para extender la funcionalidad de la API `StatisticsResource`, ya que asegura que esta herede métodos y propiedades fundamentales para la gestión de los recursos. Entre las funcionalidades heredadas se encuentran:
    - **Manejo de Seguridad:** `BaseResource` utiliza el contexto de seguridad (`SecurityContext`) proporcionado por Jakarta RESTful Web Services para identificar al usuario que realiza la solicitud. El método `getUserId()` accede al `UserPrincipal` actual, que representa al usuario autenticado, y extrae su ID de usuario.

Esta funcionalidad es esencial para verificar permisos y asegurar que solo usuarios autorizados puedan acceder a recursos específicos.

  - **Inyección de Dependencias:** La clase `BaseResource` utiliza la anotación `@Inject` para inyectar instancias de `Storage` y `PermissionsService`, que son componentes fundamentales en el manejo de datos y permisos dentro del sistema Traccar. El componente `Storage` se encarga de las operaciones de almacenamiento y recuperación de datos, mientras que `PermissionsService` se encarga de verificar que el usuario tenga los permisos necesarios para realizar determinadas acciones.
  - **Simplificación de Código:** Al centralizar estas funcionalidades en una clase base, `BaseResource` reduce la repetición de código y facilita la implementación de nuevas APIs dentro del sistema Traccar. Cualquier API que extienda `BaseResource` tendrá acceso directo a las funciones de seguridad y almacenamiento, lo que simplifica el desarrollo y el mantenimiento de la API.

- **Implementación del Código:** A continuación, se presenta el código de la clase `BaseResource`:

```
/*
 * Copyright 2015 - 2022 Anton Tananaev (
     anton@traccar.org)
 *
 * Licensed under the Apache License,
     Version 2.0 (the "License");
 * you may not use this file except in
     compliance with the License.
 * You may obtain a copy of the License at
 *
 *    http://www.apache.org/licenses/LICENSE
     -2.0
 *
 * Unless required by applicable law or
     agreed to in writing, software
 * distributed under the License is
     distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY
     KIND, either express or implied.
 * See the License for the specific language
      governing permissions and
 * limitations under the License.
 */
package org.traccar.api;

import org.traccar.api.security.
    PermissionsService;
import org.traccar.api.security.
    UserPrincipal;
import org.traccar.storage.Storage;

import jakarta.inject.Inject;
import jakarta.ws.rs.core.Context;
import jakarta.ws.rs.core.SecurityContext;

public class BaseResource {

   @Context
   private SecurityContext securityContext;

   @Inject
   protected Storage storage;

   @Inject
   protected PermissionsService
       permissionsService;

   protected long getUserId() {
      UserPrincipal principal = (
          UserPrincipal) securityContext.
          getUserPrincipal();
      if (principal != null) {
         return principal.getUserId();
      }
      return 0;
   }

}
```

Como se puede observar en el código, `BaseResource` juega un papel fundamental en la infraestructura del sistema Traccar, proporcionando un marco sólido para la construcción de APIs seguras y eficientes.

- **`import org.traccar.model.Statistics;`**
  - **Descripción:** Esta importación trae la clase `Statistics`, que representa el modelo de datos asociado a las estadísticas que maneja el sistema Traccar. Un modelo en el contexto de un sistema de software es una representación de los datos que la aplicación manipula, y en este caso, `Statistics` captura diversos aspectos del rendimiento del sistema Traccar, tales como el número de usuarios activos, dispositivos activos, solicitudes, y otros indicadores clave.
  - **Función en la API:** La clase `Statistics` es fundamental para que la API pueda crear y manipular objetos de tipo `Statistics`, los cuales son las entidades que se retornan como respuesta a las solicitudes `GET` realizadas a esta API. Cada objeto `Statistics` encapsula datos relevantes sobre el estado del sistema en un momento determinado, permitiendo a los administradores del sistema realizar análisis y monitoreo efectivos.
  - **Implementación del Código:** A continuación, se presenta el código de la clase `Statistics`:

```
/*
 * Copyright 2016 - 2017 Anton Tananaev (
     anton@traccar.org)
 *
 * Licensed under the Apache License,
     Version 2.0 (the "License");
 * you may not use this file except in
     compliance with the License.
 * You may obtain a copy of the License at
 *
 *    http://www.apache.org/licenses/LICENSE
     -2.0
 *
 * Unless required by applicable law or
     agreed to in writing, software
 * distributed under the License is
     distributed on an "AS IS" BASIS,
```

```
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY
     KIND, either express or implied.
 * See the License for the specific language
     governing permissions and
 * limitations under the License.
 */
package org.traccar.model;

import org.traccar.storage.StorageName;

import java.util.Date;
import java.util.Map;

@StorageName("tc_statistics")
public class Statistics extends
    ExtendedModel {

   private Date captureTime;

   public Date getCaptureTime() {
      return captureTime;
   }

   public void setCaptureTime(Date
       captureTime) {
      this.captureTime = captureTime;
   }

   private int activeUsers;

   public int getActiveUsers() {
      return activeUsers;
   }

   public void setActiveUsers(int
       activeUsers) {
      this.activeUsers = activeUsers;
   }

   private int activeDevices;

   public int getActiveDevices() {
      return activeDevices;
   }

   public void setActiveDevices(int
       activeDevices) {
      this.activeDevices = activeDevices;
   }

   private int requests;

   public int getRequests() {
      return requests;
   }

   public void setRequests(int requests) {
      this.requests = requests;
   }

   private int messagesReceived;

   public int getMessagesReceived() {
      return messagesReceived;
   }

   public void setMessagesReceived(int
       messagesReceived) {
      this.messagesReceived =
          messagesReceived;
   }

   private int messagesStored;

   public int getMessagesStored() {
      return messagesStored;
   }

   public void setMessagesStored(int
       messagesStored) {
      this.messagesStored = messagesStored;
   }

   private int mailSent;

   public int getMailSent() {
      return mailSent;
   }

   public void setMailSent(int mailSent) {
      this.mailSent = mailSent;
   }

   private int smsSent;

   public int getSmsSent() {
      return smsSent;
   }

   public void setSmsSent(int smsSent) {
      this.smsSent = smsSent;
   }

   private int geocoderRequests;

   public int getGeocoderRequests() {
      return geocoderRequests;
   }

   public void setGeocoderRequests(int
       geocoderRequests) {
      this.geocoderRequests =
          geocoderRequests;
   }

   private int geolocationRequests;

   public int getGeolocationRequests() {
      return geolocationRequests;
   }

   public void setGeolocationRequests(int
       geolocationRequests) {
      this.geolocationRequests =
          geolocationRequests;
   }

   private Map<String, Integer> protocols;

   public Map<String, Integer> getProtocols
       () {
      return protocols;
   }

   public void setProtocols(Map<String,
       Integer> protocols) {
      this.protocols = protocols;
   }

}
```

– **Detalles del Código:** La clase `Statistics` es un modelo que representa diferentes métricas del sistema Traccar. Aquí se explican los atributos más relevantes:

  * **captureTime**: Este atributo de tipo `Date` registra

el momento en que se capturaron las estadísticas. Es crucial para realizar análisis temporales y entender el comportamiento del sistema en diferentes periodos.

  - **activeUsers**: Este atributo de tipo `int` almacena el número de usuarios activos en el sistema en el momento de la captura. Es una métrica clave para evaluar la carga de usuarios y la utilización del sistema.
  - **activeDevices**: Similar a `activeUsers`, este atributo guarda el número de dispositivos activos. Esta métrica es útil para entender cuántos dispositivos están siendo rastreados simultáneamente.
  - **requests**: Este atributo cuenta el número de solicitudes realizadas al sistema, lo que ayuda a medir la actividad y demanda sobre la infraestructura del sistema.
  - **messagesReceived** y **messagesStored**: Estos atributos registran el número de mensajes recibidos y almacenados, respectivamente. Son importantes para evaluar la capacidad de procesamiento de mensajes y la eficacia del sistema en la gestión de la comunicación.
  - **mailSent** y **smsSent**: Guardan el número de correos electrónicos y SMS enviados por el sistema. Estas métricas son importantes para verificar el correcto funcionamiento de las notificaciones en el sistema.
  - **geocoderRequests** y **geolocationRequests**: Estos atributos almacenan el número de solicitudes al servicio de geocodificación y geolocalización, respectivamente. Permiten evaluar la precisión y la demanda de estos servicios en la plataforma.
  - **protocols**: Este es un mapa (`Map<String, Integer>`) que guarda estadísticas sobre diferentes protocolos utilizados en el sistema, cada uno con su correspondiente contador. Esto es útil para entender cuáles son los protocolos más utilizados y cómo afecta al rendimiento del sistema.

- **`import org.traccar.storage.StorageException;`**
  - **Descripción:** La clase `StorageException` se encarga de manejar las excepciones que pueden ocurrir durante las operaciones de almacenamiento de datos en el sistema Traccar. En un sistema complejo como Traccar, que depende en gran medida del almacenamiento y recuperación de datos en bases de datos, es crucial tener un manejo robusto de errores que puedan surgir durante estas operaciones. Las excepciones permiten que el sistema detecte y maneje problemas de manera controlada, en lugar de fallar abruptamente.
  - **Función en la API:** En la API `StatisticsResource`, `StorageException` se utiliza en el método `get` para manejar cualquier error que pueda surgir al interactuar con la base de datos. Esto asegura que si ocurre un problema durante la consulta de estadísticas, la excepción será capturada y manejada adecuadamente, permitiendo a la API responder con un mensaje de error controlado en lugar de interrumpir el servicio. Este manejo de excepciones es esencial para mantener la fiabilidad y la disponibilidad del servicio, incluso en situaciones de fallo.
  - **Implementación del Código:** A continuación, se presenta el código de la clase `StorageException`:

```
/*
 * Copyright 2022 Anton Tananaev (
     anton@traccar.org)
 *
 * Licensed under the Apache License,
     Version 2.0 (the "License");
 * you may not use this file except in
     compliance with the License.
 * You may obtain a copy of the License at
 *
 *    http://www.apache.org/licenses/LICENSE
     -2.0
 *
 * Unless required by applicable law or
     agreed to in writing, software
 * distributed under the License is
     distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY
     KIND, either express or implied.
 * See the License for the specific language
      governing permissions and
 * limitations under the License.
 */
package org.traccar.storage;

public class StorageException extends
    Exception {

   public StorageException(String message) {
      super(message);
   }

   public StorageException(Throwable cause)
       {
      super(cause);
   }

   public StorageException(String message,
       Throwable cause) {
      super(message, cause);
   }

}
```

  - **Detalles del Código:** La clase `StorageException` extiende `Exception`, lo que la convierte en una excepción verificable que debe ser capturada o declarada en cualquier método que pueda lanzarla. Aquí se explican los constructores disponibles en la clase:
    - **`StorageException(String message)`:** Este constructor permite crear una `StorageException` con un mensaje de error específico. Es útil para proporcionar información detallada sobre el error ocurrido, lo

que facilita el diagnóstico y la corrección del problema.

  - **`StorageException(Throwable cause)`:** Este constructor crea una `StorageException` basada en otra excepción (`Throwable`) que fue la causa original del error. Esto permite encadenar excepciones, preservando el contexto completo del error que ocurrió en niveles inferiores de la aplicación.
  - **`StorageException(String message, Throwable cause)`:** Este constructor combina las funcionalidades de los dos anteriores, permitiendo crear una excepción con un mensaje específico y una causa subyacente. Esto es particularmente útil cuando se desea informar tanto del contexto del error como de su origen exacto.

- **Importancia en el Contexto del Sistema:** La clase `StorageException` es fundamental para el manejo de errores relacionados con el almacenamiento en el sistema Traccar. Dado que el sistema depende de operaciones de base de datos para almacenar y recuperar datos de seguimiento, cualquier fallo en estas operaciones podría tener consecuencias graves, como la pérdida de datos o la interrupción del servicio. `StorageException` permite que estos errores sean detectados, informados y manejados de manera que el impacto en el usuario final sea mínimo, manteniendo la integridad y la disponibilidad del sistema.

- **`import org.traccar.storage.query.Columns;`**
  - **Descripción:** La clase `Columns` se utiliza para especificar las columnas que se desean seleccionar en una consulta a la base de datos. En un sistema como Traccar, donde se manejan grandes volúmenes de datos, es esencial poder controlar qué columnas se recuperan en una consulta para optimizar el rendimiento y reducir el uso innecesario de recursos. `Columns` permite al desarrollador definir explícitamente qué columnas deben incluirse o excluirse en los resultados de la consulta.
  - **Función en la API:** En la API `StatisticsResource`, la clase `Columns` permite a la API definir que se deben seleccionar todas las columnas de la tabla de estadísticas al realizar la consulta. Esto se logra mediante la instrucción `new Columns.All()`, que asegura que la API recupere todos los datos disponibles en la tabla correspondiente, permitiendo a los administradores obtener un conjunto completo de estadísticas sin omitir ninguna información relevante.
  - **Implementación del Código:** A continuación, se presenta el código de la clase `Columns`:

```
/*
 * Copyright 2022 Anton Tananaev (
    anton@traccar.org)
 *
 * Licensed under the Apache License,
    Version 2.0 (the "License");
 * you may not use this file except in
    compliance with the License.
 * You may obtain a copy of the License at
 *
 *    http://www.apache.org/licenses/LICENSE
    -2.0
 *
 * Unless required by applicable law or
    agreed to in writing, software
 * distributed under the License is
    distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY
    KIND, either express or implied.
 * See the License for the specific language
     governing permissions and
 * limitations under the License.
 */
package org.traccar.storage.query;

import org.traccar.storage.QueryIgnore;

import java.beans.Introspector;
import java.lang.reflect.Method;
import java.util.Arrays;
import java.util.LinkedList;
import java.util.List;
import java.util.Set;
import java.util.stream.Collectors;

public abstract class Columns {

   public abstract List<String> getColumns(
       Class<?> clazz, String type);

   protected List<String> getAllColumns(
       Class<?> clazz, String type) {
      List<String> columns = new LinkedList
          <>();
      Method[] methods = clazz.getMethods();
      for (Method method : methods) {
         int parameterCount = type.equals("
             set") ? 1 : 0;
         if (method.getName().startsWith(
             type) && method.
             getParameterTypes().length ==
             parameterCount
               && !method.
                   isAnnotationPresent(
                   QueryIgnore.class)
               && !method.getName().equals("
                   getClass")) {
            columns.add(Introspector.
                decapitalize(method.getName
                ().substring(3)));
         }
      }
      return columns;
   }

   public static class All extends Columns {
      @Override
      public List<String> getColumns(Class
          <?> clazz, String type) {
         return getAllColumns(clazz, type);
      }
   }

   public static class Include extends
       Columns {
```

```
        private final List<String> columns;

        public Include(String... columns) {
            this.columns = Arrays.stream(
                columns).collect(Collectors.
                toList());
        }

        @Override
        public List<String> getColumns(Class
            <?> clazz, String type) {
            return columns;
        }
    }

    public static class Exclude extends
        Columns {
        private final Set<String> columns;

        public Exclude(String... columns) {
            this.columns = Arrays.stream(
                columns).collect(Collectors.
                toSet());
        }

        @Override
        public List<String> getColumns(Class
            <?> clazz, String type) {
            return getAllColumns(clazz, type).
                stream()
                  .filter(column -> !columns.
                      contains(column))
                  .collect(Collectors.toList())
                      ;
        }
    }

}
```

- **Detalles del Código:** La clase `Columns` es una abstracción que facilita la especificación de las columnas a seleccionar en una consulta. Aquí se explican los componentes más relevantes:
  - **Método `getAllColumns(Class<?> clazz, String type)`:** Este método protegido devuelve una lista de todas las columnas correspondientes a los métodos de acceso (getters) o modificación (setters) en la clase especificada por `clazz`. Se filtran los métodos que están anotados con `@QueryIgnore`, y aquellos que no son relevantes para la consulta, como `getClass`. Este enfoque garantiza que solo las columnas significativas se incluyan en los resultados de la consulta.
  - **Clase `All`:** Esta clase interna extiende `Columns` y sobrescribe el método `getColumns` para devolver todas las columnas disponibles en la clase especificada. Esta es la clase utilizada en la API `StatisticsResource` cuando se desea recuperar toda la información disponible sin omitir ningún campo.
  - **Clase `Include`:** Permite especificar explícitamente qué columnas deben incluirse en la consulta. Al construir un objeto de esta clase con los nombres de las columnas deseadas, se puede limitar la consulta solo a los datos más relevantes, optimizando así el rendimiento y reduciendo el volumen de datos transferidos.
  - **Clase `Exclude`:** Esta clase funciona de manera opuesta a `Include`, permitiendo excluir columnas específicas de la consulta. Esto es útil cuando se desean omitir ciertas columnas que no son necesarias para una operación particular, mientras se recuperan todas las demás.
- **Importancia en el Contexto del Sistema:** La flexibilidad proporcionada por la clase `Columns` es crucial en un sistema como Traccar, donde diferentes consultas pueden requerir diferentes subconjuntos de datos. Al permitir la inclusión o exclusión de columnas según sea necesario, `Columns` optimiza el rendimiento del sistema y proporciona a los desarrolladores un control granular sobre los datos que se recuperan en cada operación de consulta.

- **`import org.traccar.storage.query.Condition;`**
  - **Descripción:** La clase `Condition` es una interfaz que se utiliza para definir las condiciones que deben cumplirse en una consulta a la base de datos. En sistemas como Traccar, donde las consultas a la base de datos pueden ser complejas y específicas, `Condition` proporciona un marco estructurado para crear estas condiciones de manera modular y reutilizable.
  - **Función en la API:** En la API `StatisticsResource`, la clase `Condition` es crucial para filtrar los resultados basados en un rango de fechas. Esto se logra utilizando la condición `Condition.Between`, que asegura que solo se devuelvan estadísticas dentro del intervalo especificado por el usuario. De esta manera, se optimiza la consulta y se proporciona al usuario únicamente la información relevante.
  - **Implementación del Código:** A continuación, se presenta el código de la interfaz `Condition` y sus clases internas:

```
/*
 * Copyright 2022 Anton Tananaev (
     anton@traccar.org)
 *
 * Licensed under the Apache License,
     Version 2.0 (the "License");
 * you may not use this file except in
     compliance with the License.
 * You may obtain a copy of the License at
 *
 *    http://www.apache.org/licenses/LICENSE
     -2.0
 *
 * Unless required by applicable law or
     agreed to in writing, software
 * distributed under the License is
     distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY
     KIND, either express or implied.
 * See the License for the specific language
      governing permissions and
```

```
 * limitations under the License.
 */
package org.traccar.storage.query;

import org.traccar.model.GroupedModel;

import java.util.List;

public interface Condition {

   static Condition merge(List<Condition>
       conditions) {
      Condition result = null;
      var iterator = conditions.iterator();
      if (iterator.hasNext()) {
         result = iterator.next();
         while (iterator.hasNext()) {
            result = new Condition.And(
                result, iterator.next());
         }
      }
      return result;
   }

   class Equals extends Compare {
      public Equals(String column, Object
          value) {
         super(column, "=", column, value);
      }
   }

   class Compare implements Condition {
      private final String column;
      private final String operator;
      private final String variable;
      private final Object value;

      public Compare(String column, String
          operator, String variable, Object
          value) {
         this.column = column;
         this.operator = operator;
         this.variable = variable;
         this.value = value;
      }

      public String getColumn() {
         return column;
      }

      public String getOperator() {
         return operator;
      }

      public String getVariable() {
         return variable;
      }

      public Object getValue() {
         return value;
      }
   }

   class Between implements Condition {
      private final String column;
      private final String fromVariable;
      private final Object fromValue;
      private final String toVariable;
      private final Object toValue;

      public Between(String column, String
          fromVariable, Object fromValue,
          String toVariable, Object toValue)
           {
         this.column = column;
         this.fromVariable = fromVariable;
         this.fromValue = fromValue;
         this.toVariable = toVariable;
         this.toValue = toValue;
      }

      public String getColumn() {
         return column;
      }

      public String getFromVariable() {
         return fromVariable;
      }

      public Object getFromValue() {
         return fromValue;
      }

      public String getToVariable() {
         return toVariable;
      }

      public Object getToValue() {
         return toValue;
      }
   }

   class Or extends Binary {
      public Or(Condition first, Condition
          second) {
         super(first, second, "OR");
      }
   }

   class And extends Binary {
      public And(Condition first, Condition
          second) {
         super(first, second, "AND");
      }
   }

   class Binary implements Condition {
      private final Condition first;
      private final Condition second;
      private final String operator;

      public Binary(Condition first,
          Condition second, String operator)
           {
         this.first = first;
         this.second = second;
         this.operator = operator;
      }

      public Condition getFirst() {
         return first;
      }

      public Condition getSecond() {
         return second;
      }

      public String getOperator() {
         return operator;
      }
   }

   class Permission implements Condition {
      private final Class<?> ownerClass;
      private final long ownerId;
      private final Class<?> propertyClass;
      private final long propertyId;
      private final boolean excludeGroups;
```

```
    private Permission(
          Class<?> ownerClass, long
              ownerId, Class<?>
              propertyClass, long
              propertyId, boolean
              excludeGroups) {
       this.ownerClass = ownerClass;
       this.ownerId = ownerId;
       this.propertyClass = propertyClass;
       this.propertyId = propertyId;
       this.excludeGroups = excludeGroups;
    }

    public Permission(Class<?> ownerClass,
         long ownerId, Class<?>
        propertyClass) {
       this(ownerClass, ownerId,
           propertyClass, 0, false);
    }

    public Permission(Class<?> ownerClass,
         Class<?> propertyClass, long
        propertyId) {
       this(ownerClass, 0, propertyClass,
           propertyId, false);
    }

    public Permission excludeGroups() {
       return new Permission(this.
           ownerClass, this.ownerId, this.
           propertyClass, this.propertyId,
            true);
    }

    public Class<?> getOwnerClass() {
       return ownerClass;
    }

    public long getOwnerId() {
       return ownerId;
    }

    public Class<?> getPropertyClass() {
       return propertyClass;
    }

    public long getPropertyId() {
       return propertyId;
    }

    public boolean getIncludeGroups() {
       boolean ownerGroupModel =
           GroupedModel.class.
           isAssignableFrom(ownerClass);
       boolean propertyGroupModel =
           GroupedModel.class.
           isAssignableFrom(propertyClass)
           ;
       return (ownerGroupModel ||
           propertyGroupModel) && !
           excludeGroups;
    }
}

class LatestPositions implements
    Condition {
    private final long deviceId;

    public LatestPositions(long deviceId)
        {
       this.deviceId = deviceId;
    }

    public LatestPositions() {
       this(0);
    }

    public long getDeviceId() {
       return deviceId;
    }
  }

}
```

– **Detalles del Código:** La interfaz `Condition` define la estructura básica para las condiciones de las consultas y proporciona varias implementaciones concretas que permiten la construcción de condiciones específicas. Aquí se explican los componentes más relevantes:
  * **Método `merge(List<Condition> conditions)`:** Este método estático toma una lista de condiciones y las combina en una sola utilizando la operación lógica AND. Esto es útil cuando se desean aplicar múltiples condiciones simultáneamente en una consulta.
  * **Clase `Between`:** Esta clase se utiliza para crear una condición de tipo BETWEEN, que es comúnmente utilizada para filtrar resultados dentro de un rango. En la API `StatisticsResource`, se utiliza para asegurarse de que las estadísticas devueltas están dentro del rango de fechas especificado.
  * **Clase `Equals`:** Esta clase es una implementación concreta de una comparación de igualdad (=) entre una columna y un valor dado. Es útil para filtrar resultados que coinciden exactamente con un valor específico.
  * **Clase `And` y `Or`:** Estas clases permiten combinar dos condiciones utilizando las operaciones lógicas AND y OR, respectivamente. Esto es esencial para construir consultas más complejas que requieren múltiples criterios de filtrado.
  * **Clase `Permission`:** Esta clase es particularmente importante en el contexto de seguridad, ya que permite crear condiciones basadas en los permisos del usuario. Puede filtrar resultados según los derechos de acceso, asegurando que solo los datos permitidos sean accesibles para un usuario en particular.
  * **Clase `LatestPositions`:** Aunque no se utiliza en la API `StatisticsResource`, esta clase proporciona un ejemplo de una condición específica del dominio de Traccar, donde se filtran las posiciones más recientes de un dispositivo dado.
– **Importancia en el Contexto del Sistema:** La interfaz `Condition` y sus implementaciones son fundamentales para construir consultas dinámicas y flexibles en el sistema Traccar. Permiten que la API defina exactamente qué datos deben ser recupera-

dos y bajo qué condiciones, optimizando tanto el rendimiento de las consultas como la relevancia de los datos devueltos. Al proporcionar una estructura clara para las condiciones, `Condition` facilita la implementación de consultas complejas de manera modular y reutilizable.

- **`import org.traccar.storage.query.Order;`**
  - **Descripción:** La clase `Order` se utiliza para especificar el orden en que se deben devolver los resultados de una consulta a la base de datos. En sistemas de gestión de bases de datos, el ordenamiento de los resultados es una operación común y esencial, especialmente cuando se trabaja con conjuntos de datos grandes y se requiere una organización específica de los resultados, como en orden cronológico o alfabético.
  - **Función en la API:** En la API `StatisticsResource`, la clase `Order` se utiliza para ordenar las estadísticas por la columna `captureTime`, lo que asegura que los resultados se presenten en orden cronológico. Esto es crucial para los usuarios que necesitan analizar las estadísticas en secuencia temporal, permitiéndoles ver cómo evolucionan los datos a lo largo del tiempo.
  - **Implementación del Código:** A continuación, se presenta el código de la clase `Order`:

```
/*
 * Copyright 2022 Anton Tananaev (
     anton@traccar.org)
 *
 * Licensed under the Apache License,
     Version 2.0 (the "License");
 * you may not use this file except in
     compliance with the License.
 * You may obtain a copy of the License at
 *
 *    http://www.apache.org/licenses/LICENSE
     -2.0
 *
 * Unless required by applicable law or
     agreed to in writing, software
 * distributed under the License is
     distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY
     KIND, either express or implied.
 * See the License for the specific language
      governing permissions and
 * limitations under the License.
 */
package org.traccar.storage.query;

public class Order {

   private final String column;
   private final boolean descending;
   private final int limit;

   public Order(String column) {
      this(column, false, 0);
   }

   public Order(String column, boolean
       descending, int limit) {
      this.column = column;
      this.descending = descending;
      this.limit = limit;
   }

   public String getColumn() {
      return column;
   }

   public boolean getDescending() {
      return descending;
   }

   public int getLimit() {
      return limit;
   }

}
```

  - **Detalles del Código:** La clase `Order` encapsula la información necesaria para ordenar los resultados de una consulta en la base de datos. Aquí se explican los componentes más relevantes:
    - **Atributo `column`:** Este atributo almacena el nombre de la columna por la cual se debe ordenar la consulta. En el caso de la API `StatisticsResource`, se utiliza `captureTime`, lo que asegura que las estadísticas se ordenen cronológicamente.
    - **Atributo `descending`:** Este atributo booleano indica si el orden debe ser descendente. Si es `false`, los resultados se ordenan de manera ascendente; si es `true`, se ordenan de manera descendente. Esto es útil para ordenar datos en orden inverso, como de la fecha más reciente a la más antigua.
    - **Atributo `limit`:** Este atributo define un límite en el número de resultados que se deben devolver. Si el valor es mayor que cero, se limita el número de filas devueltas por la consulta. Esto es útil cuando solo se desea recuperar un subconjunto de los datos, como las estadísticas más recientes o más relevantes.
    - **Constructor `Order(String column)`:** Este constructor permite crear un objeto `Order` con el nombre de la columna especificada y valores predeterminados para los atributos `descending` y `limit` (es decir, orden ascendente y sin límite en el número de resultados).
    - **Constructor `Order(String column, boolean descending, int limit)`:** Este constructor más avanzado permite especificar no solo la columna, sino también si el orden debe ser descendente y si debe aplicarse un límite al número de resultados devueltos.
  - **Importancia en el Contexto del Sistema:** La clase `Order` es esencial para controlar cómo se presentan los resultados de las consultas en el sistema Traccar. Al permitir que los desarrolladores especifiquen tanto la columna de ordenación como la dirección del orden y el límite de resultados, `Order` propor-

ciona una flexibilidad crucial en la manipulación de grandes conjuntos de datos, asegurando que los usuarios obtengan los resultados en el formato y orden que mejor se adapte a sus necesidades.

- **`import org.traccar.storage.query.Request;`**
  - **Descripción:** La clase `Request` se utiliza para construir y ejecutar consultas a la base de datos que involucran selección de columnas, condiciones, y ordenamiento. Es una clase central en el sistema Traccar que permite combinar estos tres aspectos clave de una consulta en un solo objeto coherente, facilitando la interacción con la base de datos de manera flexible y eficiente.
  - **Función en la API:** En la API `StatisticsResource`, la clase `Request` es fundamental, ya que encapsula todos los aspectos de la consulta a la base de datos en un solo objeto. Esto incluye las columnas que deben ser seleccionadas, las condiciones que deben cumplirse y el orden en el que los resultados deben ser devueltos. Al reunir estos elementos en un único objeto `Request`, se simplifica la lógica de la API y se asegura que las consultas se realicen de manera consistente según los parámetros definidos por el usuario.
  - **Implementación del Código:** A continuación, se presenta el código de la clase `Request`:

```
/*
 * Copyright 2022 Anton Tananaev (
    anton@traccar.org)
 *
 * Licensed under the Apache License,
    Version 2.0 (the "License");
 * you may not use this file except in
    compliance with the License.
 * You may obtain a copy of the License at
 *
 *    http://www.apache.org/licenses/LICENSE
    -2.0
 *
 * Unless required by applicable law or
    agreed to in writing, software
 * distributed under the License is
    distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY
    KIND, either express or implied.
 * See the License for the specific language
     governing permissions and
 * limitations under the License.
 */
package org.traccar.storage.query;

public class Request {

   private final Columns columns;
   private final Condition condition;
   private final Order order;

   public Request(Columns columns) {
      this(columns, null, null);
   }

   public Request(Condition condition) {
      this(null, condition, null);
   }

   public Request(Columns columns, Condition
       condition) {
      this(columns, condition, null);
   }

   public Request(Columns columns, Order
       order) {
      this(columns, null, order);
   }

   public Request(Columns columns, Condition
       condition, Order order) {
      this.columns = columns;
      this.condition = condition;
      this.order = order;
   }

   public Columns getColumns() {
      return columns;
   }

   public Condition getCondition() {
      return condition;
   }

   public Order getOrder() {
      return order;
   }

}
```

  - **Detalles del Código:** La clase `Request` permite construir consultas complejas de manera sencilla y modular, combinando selección de columnas, condiciones de filtrado, y criterios de ordenamiento en una sola estructura. Aquí se explican los componentes más relevantes:
    - **Atributo `columns`:** Este atributo de tipo `Columns` define las columnas que se deben seleccionar en la consulta. Puede ser un objeto que especifica todas las columnas (`Columns.All`) o un subconjunto específico. Si se pasa `null`, no se aplicará un filtro de columnas.
    - **Atributo `condition`:** Este atributo de tipo `Condition` define las condiciones que deben cumplirse para que un registro sea incluido en el resultado de la consulta. Esto puede incluir condiciones como `Condition.Between` para filtrar datos por un rango de fechas. Si se pasa `null`, no se aplicará ninguna condición.
    - **Atributo `order`:** Este atributo de tipo `Order` especifica el orden en que se deben devolver los resultados, como por ejemplo, ordenados por fecha (`Order("captureTime")`). Si se pasa `null`, los resultados se devolverán en el orden predeterminado de la base de datos.
    - **Constructores:** La clase `Request` incluye varios constructores que permiten crear una solicitud con diferentes combinaciones de columnas, condiciones y órdenes:
      - **Constructor `Request(Columns columns)`:** Crea una solicitud solo con

la selección de columnas.
    - **Constructor `Request(Condition condition)`:** Crea una solicitud solo con las condiciones de filtrado.
    - **Constructor `Request(Columns columns, Condition condition)`:** Crea una solicitud con selección de columnas y condiciones.
    - **Constructor `Request(Columns columns, Order order)`:** Crea una solicitud con selección de columnas y ordenamiento.
    - **Constructor `Request(Columns columns, Condition condition, Order order)`:** Crea una solicitud con selección de columnas, condiciones y ordenamiento.
  - **Métodos `getColumns()`, `getCondition()` y `getOrder()`:** Estos métodos devuelven los respectivos atributos, permitiendo acceder a los detalles de la solicitud cuando se ejecuta la consulta en la base de datos.
- **Importancia en el Contexto del Sistema:** La clase `Request` es vital para el funcionamiento eficiente de las consultas en el sistema Traccar. Al encapsular la selección de columnas, las condiciones y el orden en un solo objeto, `Request` permite una construcción de consultas flexible y reutilizable, reduciendo la complejidad en el código de la API y asegurando que las consultas a la base de datos se ejecuten de manera coherente y optimizada.

- **`import jakarta.ws.rs.Consumes;`**
  - **Descripción:** Esta anotación indica que la API puede consumir datos en un formato específico, en este caso `MediaType.APPLICATION_JSON`.
  - `Función en la API`: Es importante para que la API sepa que los datos de entrada que procesa están en formato JSON, asegurando una correcta interpretación de las solicitudes.
- **`import jakarta.ws.rs.GET;`**
  - **Descripción:** Anotación que define que un método es accesible mediante solicitudes HTTP GET.
  - **Función en la API:** Esencial para indicar que el método `get` en `StatisticsResource` se invoca cuando se realiza una solicitud GET al endpoint correspondiente.
- **`import jakarta.ws.rs.Path;`**
  - **Descripción:** Anotación que define la ruta de acceso al recurso, en este caso `statistics`.
  - **Función en la API:** Permite que la clase `StatisticsResource` sea accesible en la URL definida, permitiendo a los clientes interactuar con el recurso de estadísticas.
- **`import jakarta.ws.rs.Produces;`**
  - **Descripción:** Anotación que especifica el tipo de contenido que la API produce en la respuesta, en este caso `MediaType.APPLICATION_JSON`.
  - **Función en la API:** Es necesaria para garantizar que las respuestas de la API estén en formato JSON, que es el formato estándar esperado por los clientes.
- **`import jakarta.ws.rs.QueryParam;`**
  - **Descripción:** Anotación que indica que un parámetro del método es un parámetro de consulta en la URL.
  - **Función en la API:** Se utiliza para recibir los parámetros `from` y `to` en el método `get`, los cuales determinan el rango de fechas para la consulta de estadísticas.
- **`import jakarta.ws.rs.core.MediaType;`**
  - **Descripción:** Clase que contiene constantes para los tipos de medios MIME, como `APPLICATION_JSON`.
  - **Función en la API:** Es fundamental para especificar que la API consumirá y producirá datos en formato JSON, lo que garantiza la correcta interpretación de solicitudes y respuestas.
- **`import java.util.Collection;`**
  - **Descripción:** Clase de la biblioteca estándar de Java que representa un grupo de elementos.
  - **Función en la API:** Es utilizada como el tipo de retorno del método `get`, permitiendo que la API devuelva una colección de objetos `Statistics`.
- **`import java.util.Date;`**
  - **Descripción:** Clase de la biblioteca estándar de Java que representa una fecha y hora específicas.
  - **Función en la API:** Se utiliza para representar los parámetros de fecha `from` y `to`, que son utilizados para filtrar las estadísticas dentro de un rango de fechas específico.

D. Documentación Bloque de Código

A continuación, se presenta una explicación detallada de cada línea del bloque de código:

- `@Path("statistics")`: Define la ruta del recurso de la API. En este caso, la ruta será `/statistics`. Cualquier solicitud a esta URL será manejada por esta clase.
- `@Produces(MediaType.APPLICATION_JSON)`: Indica que las respuestas generadas por los métodos de esta clase serán en formato JSON, lo que es estándar para APIs RESTful.
- `@Consumes(MediaType.APPLICATION_JSON)`: Especifica que esta API consume (acepta) solicitudes en formato JSON. Esto es relevante para solicitudes POST, PUT, etc., aunque en este caso es un GET.
- `public class StatisticsResource extends BaseRes` Define la clase `StatisticsResource` que extiende `BaseResource`. Esto significa que hereda funcionalidades básicas de `BaseResource`, que

podrían incluir gestión de permisos, manejo de excepciones, entre otros.

- `@GET`: Anotación que indica que el método `get()` responderá a solicitudes HTTP GET. Este es el método que se utilizará para obtener recursos, en este caso, estadísticas.
- `public Collection<Statistics> get(`: Declaración del método `get`, que retorna una colección de objetos `Statistics`. Este método acepta dos parámetros de tipo `Date`, `from` y `to`, que definen el rango de fechas para la consulta.
- `@QueryParam("from") Date from, @QueryParam("to") Date to)`: Los parámetros de consulta `from` y `to` se extraen de la URL de la solicitud. Estos parámetros se utilizan para filtrar las estadísticas por rango de fechas.
- `throws StorageException {`: Declara que el método puede lanzar una `StorageException`, que es una excepción específica para errores relacionados con el almacenamiento de datos, como consultas a la base de datos.
- `permissionsService.checkAdmin(getUserId());`: Verifica que el usuario que realiza la solicitud tiene privilegios de administrador. Si el usuario no es administrador, se podría lanzar una excepción o denegar la solicitud.
- `return storage.getObjects( Statistics.class, new Request(`: Realiza una consulta a la base de datos para obtener los objetos `Statistics` dentro del rango de fechas especificado. La consulta se construye mediante un objeto `Request`.
- `new Columns.All(),`: Indica que se deben seleccionar todas las columnas de la tabla de estadísticas en la consulta a la base de datos.
- `new Condition.Between("captureTime", "from", from, "to", to),`: Establece la condición de la consulta, especificando que se deben seleccionar las estadísticas cuya columna `captureTime` esté entre las fechas `from` y `to`.
- `new Order("captureTime")));`: Ordena los resultados de la consulta por la columna `captureTime`, lo que garantiza que las estadísticas se devuelvan en orden cronológico.
- `}`: Cierra el método `get()`.
- `}`: Cierra la clase `StatisticsResource`.

IV. Resultados

En esta sección se presentan los resultados obtenidos al documentar la API `StatisticsResource.java`, siguiendo un formato estructurado que cubre los aspectos esenciales de una API RESTful. A continuación, se detallan los pasos generales que se siguieron para documentar esta API:

A. Título de la API

El nombre o título de la API es ”API de Estadísticas”.

B. Descripción General

La API de Estadísticas permite a los administradores obtener estadísticas de la plataforma dentro de un rango de fechas especificado. Esta API está diseñada para proporcionar información valiosa sobre el uso del sistema, facilitando el análisis y la toma de decisiones.

C. URL del Endpoint

La ruta donde está disponible el recurso es `/statistics`.

D. Método HTTP

El método HTTP utilizado por la API es `GET`. Este método se usa para recuperar los datos de estadísticas almacenados en el sistema.

E. Parámetros de Entrada

La API acepta los siguientes parámetros de entrada:

- `from`: Fecha de inicio para filtrar las estadísticas. El formato del parámetro es `Date`.
- `to`: Fecha de fin para filtrar las estadísticas. El formato del parámetro es `Date`.

F. Cuerpo de la Solicitud (Request Body)

Dado que se trata de una solicitud `GET`, no se requiere un cuerpo de solicitud. Toda la información necesaria se pasa a través de los parámetros de consulta en la URL.

G. Encabezados (Headers)

Dependiendo de la configuración del servidor, es posible que se requieran encabezados adicionales, como tokens de autenticación, para acceder a esta API. Sin embargo, no se han especificado encabezados obligatorios para este ejemplo.

H. Ejemplo de Solicitud

A continuación se muestra un ejemplo de cómo realizar una solicitud a esta API:

```
GET /statistics?from=2023-01-01T00:00:00Z&to=2023-
```

I. Respuesta de la API

La respuesta de la API será en formato JSON y contendrá una colección de objetos `Statistics`. A continuación se muestra un ejemplo del formato de respuesta:

```
[
  {
    "id": 1,
    "captureTime": "2023-01-01T00:00:00Z",
    "value": 123
  },
  ...
]
```

J. Códigos de Respuesta

La API puede devolver los siguientes códigos de respuesta HTTP:

- `200 OK`: La solicitud fue exitosa y los datos fueron devueltos correctamente.
- `400 Bad Request`: Los parámetros proporcionados son incorrectos o faltantes.
- `401 Unauthorized`: El usuario no tiene los permisos necesarios para acceder a este recurso.
- `500 Internal Server Error`: Se produjo un error en el servidor al procesar la solicitud.

K. Ejemplos de Respuesta

Un ejemplo de respuesta exitosa sería:

```
[
  {
    "id": 1,
    "captureTime": "2023-01-01T00:00:00Z",
    "value": 123
  }
]
```

En caso de error, la respuesta podría ser:

```
{
  "error": "Invalid date range"
}
```

L. Notas Adicionales

Finalmente, se pueden incluir notas adicionales que proporcionen información relevante, como restricciones, límites de uso, o consideraciones especiales para el uso de esta API en diferentes entornos.

Con esta estructura, se ha documentado de manera clara y detallada la API `StatisticsResource.java`, facilitando su comprensión y uso por parte de los desarrolladores que deseen integrarla o mantenerla en el futuro.

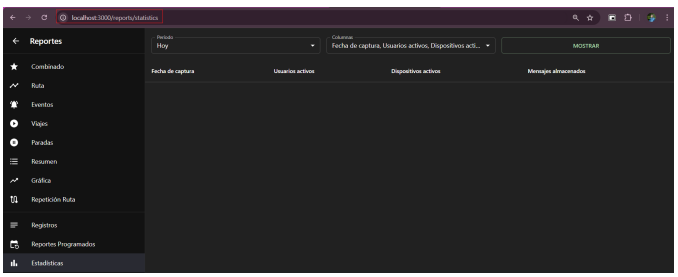

Fig. 1. Descripción de la primera imagen.

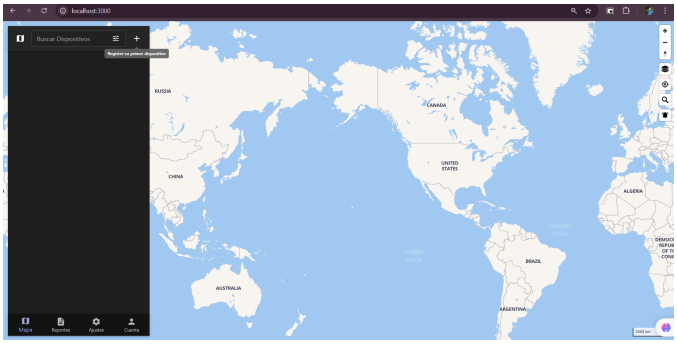

Fig. 2. Descripción de la segunda imagen.

V. Discusión y Conclusiones

La documentación de una API es una tarea esencial para garantizar que los desarrolladores puedan entender y utilizar correctamente los servicios que ésta proporciona. En este proyecto, se ha documentado la API `StatisticsResource` perteneciente al sistema Traccar, siguiendo un formato estructurado que aborda los aspectos clave de la misma.

Durante el proceso de documentación, se resaltó la importancia de incluir detalles como el título de la API, una descripción general, los métodos HTTP que se utilizan, los parámetros de entrada, y los posibles códigos de respuesta. Esta estructura no solo facilita la comprensión del funcionamiento de la API, sino que también asegura que los desarrolladores puedan integrarla sin dificultades en sus propias aplicaciones.

La API `StatisticsResource` es una herramienta fundamental para los administradores del sistema Traccar, ya que les permite obtener estadísticas detalladas sobre el rendimiento del sistema en un rango de fechas específico. La correcta documentación de esta API garantiza que los usuarios con los permisos adecuados puedan acceder a estos datos de manera eficiente y segura.

En cuanto a las conclusiones, se puede afirmar que:

- La documentación de APIs debe ser clara, completa y estructurada para facilitar su uso por parte de terceros.
- El uso de anotaciones y el manejo adecuado de los métodos HTTP son fundamentales para definir el comportamiento de las APIs RESTful.
- Es crucial incluir ejemplos prácticos de solicitudes y respuestas, así como los posibles errores que puedan surgir durante el uso de la API, para anticipar y mitigar problemas durante su implementación.
- La API `StatisticsResource` es un componente crítico en el sistema Traccar, y su documentación detallada permitirá a los administradores del sistema maximizar su utilidad y eficacia.

En resumen, la correcta documentación de la API `StatisticsResource` no solo facilita su integración y uso, sino que también contribuye a la robustez y escalabilidad del sistema Traccar, asegurando que el acceso a las estadísticas del sistema sea confiable y seguro.

REFERENCIAS

[1] A. Tananaev, "Traccar: An open source gps tracking system," in *Proceedings of the 7th International Conference on Software Development*. IEEE, 2018.

[2] ——, "Open source gps tracking: Traccar," *Journal of Software Engineering and Applications*, vol. 9, no. 7, pp. 345–355, 2016.

[3] K. Munir, Z. Anwar, and O. Mohamad, "Big data and predictive analytics in cloud computing: Challenges and opportunities," *Journal of Cloud Computing*, vol. 5, no. 1, p. 22, 2016.

[4] J. Han, J. Pei, and M. Kamber, *Data Mining: Concepts and Techniques*. Elsevier, 2012.

[5] L. Richardson, M. Amundsen, and S. Ruby, *RESTful Web APIs*. O'Reilly Media, Inc., 2016.