Docuemntación API de Position

Universidad de las Fuerzas Armadas, Belisario Quevedo, Barrio El Forastero, Latacunga, 050105, Ecuador

I. Position

A. *Descripción*

La API de Traccar Position se encarga de gestionar las operaciones relacionadas con las posiciones geográficas de los dispositivos de rastreo. Esta clase permite obtener, eliminar y exportar los datos de posición en diferentes formatos (KML, CSV, GPX).

B. *¿Cómo funciona?*

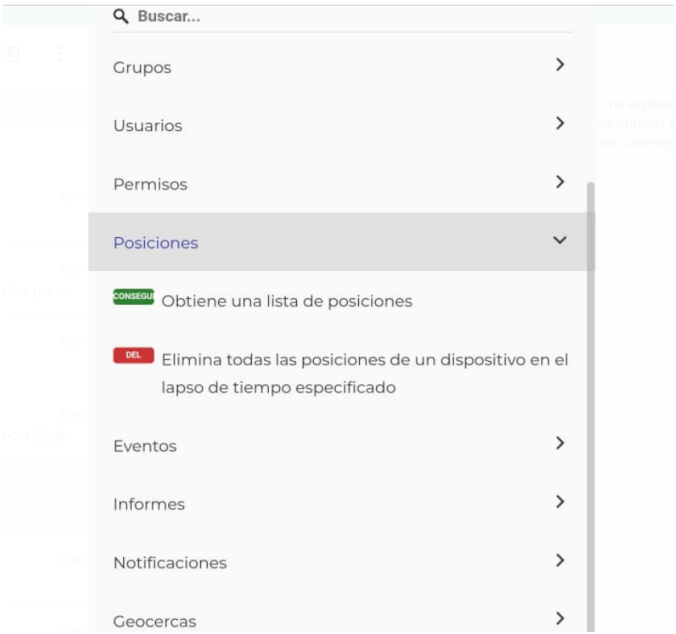

Fig. 1. Get y Delete

`PositionResource` es un recurso REST que expone diferentes endpoints para interactuar con las posiciones de los dispositivos. Los métodos principales permiten:

- **Obtener Posiciones (getJson):** Permite recuperar posiciones de un dispositivo específico dentro de un rango de tiempo o mediante IDs específicas de posición.
- **Eliminar Posiciones (remove):** Permite eliminar posiciones de un dispositivo en un rango de tiempo determinado.
- **Exportar Posiciones:**
  - **KML (getKml):** Exporta las posiciones en formato KML, utilizado comúnmente en aplicaciones como Google Earth.
  - **CSV (getCsv):** Exporta las posiciones en formato CSV, que es fácil de manipular en aplicaciones de hoja de cálculo.
  - **GPX (getGpx):** Exporta las posiciones en formato GPX, comúnmente usado en dispositivos de GPS.

C. *Funcionalidades Principales*

- **Filtrado de posiciones:** Permite filtrar posiciones por ID, dispositivo, y por rango de tiempo.
- **Permisos y restricciones:** Antes de realizar operaciones, la API verifica que el usuario tenga permisos para acceder a los datos del dispositivo y que no tenga restricciones como el modo de solo lectura.
- **Exportación:** La API permite exportar los datos de posición en diferentes formatos estándar (KML, CSV, GPX).
- **Eliminación de datos:** Se pueden eliminar datos de posición para un dispositivo específico en un rango de tiempo dado.

D. *Dependencias Principales*

- **org.traccar.api.BaseResource**: Clase base para los recursos de la API en Traccar, proporcionando funcionalidades comunes.
- **org.traccar.helper.model.PositionUtil**: Utilidad para manipular y obtener datos de posiciones.

- **org.traccar.model.Device**: Representa un dispositivo de rastreo.
- **org.traccar.model.Position**: Representa una posición geográfica registrada por un dispositivo.
- **org.traccar.model.UserRestrictions**: Clase que define las restricciones aplicables a un usuario.
- **org.traccar.reports.CsvExportProvider**: Proveedor para generar exportaciones de posiciones en formato CSV.
- **org.traccar.reports.GpxExportProvider**: Proveedor para generar exportaciones en formato GPX.
- **org.traccar.reports.KmlExportProvider**: Proveedor para generar exportaciones en formato KML.
- **org.traccar.storage.StorageException**: Excepción lanzada en caso de problemas con la base de datos o almacenamiento.
- **org.traccar.storage.query.Columns**: Utilizado para especificar las columnas a recuperar en una consulta.
- **org.traccar.storage.query.Condition**: Utilizado para definir condiciones en las consultas a la base de datos.
- **org.traccar.storage.query.Request**: Clase que encapsula una solicitud de consulta a la base de datos.

Descripción del Código - `PositionResource`

El código que has compartido pertenece a la clase `PositionResource`, que es parte de un servicio web utilizado para gestionar las posiciones de dispositivos, típicamente en sistemas de rastreo GPS como Traccar. Este servicio permite a los usuarios recuperar y eliminar datos de posiciones, así como exportarlos en varios formatos como KML, CSV y GPX. A continuación, se explica cómo funciona el código.

Dependencias:

- `KmlExportProvider`, `CsvExportProvider`, `GpxExportProvider`: Estas clases se utilizan para generar archivos de exportación en los formatos KML, CSV y GPX, respectivamente.
- `PositionUtil`: Una clase de utilidad para gestionar datos de posiciones.
- `StorageException`: Una clase de excepción que maneja errores relacionados con el almacenamiento de datos.

Métodos Clave: **GET: Recuperar Posiciones (`getJson`)**

- **Parámetros:**
  - `deviceId`: El ID del dispositivo cuyas posiciones se están solicitando.
  - `positionIds`: Una lista de IDs de posiciones específicas.
  - `from` y `to`: Rango de fechas para filtrar las posiciones.
- **Funcionalidad:**
  - Si se proporciona `positionIds`, el método recupera esas posiciones específicas.
  - Si se proporciona `deviceId`, el método recupera las posiciones para ese dispositivo dentro del rango de fechas especificado o las posiciones más recientes si no se da un rango de fechas.
  - Si no se proporciona ni `positionIds` ni `deviceId`, se recuperan las posiciones más recientes accesibles para el usuario.
- **Permisos:** El método verifica si el usuario tiene los permisos necesarios para acceder al dispositivo y a las posiciones.

DELETE: Eliminar Posiciones (`remove`)

- **Parámetros:**
  - `deviceId`: El ID del dispositivo cuyas posiciones se desean eliminar.
  - `from` y `to`: Rango de fechas para eliminar posiciones.
- **Funcionalidad:** Elimina posiciones dentro del rango de fechas especificado para el dispositivo dado.
- **Permisos:** Asegura que el usuario tenga permiso para eliminar posiciones y que su cuenta no esté restringida a solo lectura.

GET: Exportar a KML (`getKml`)

- **Parámetros:**
  - `deviceId`: El ID del dispositivo.
  - `from` y `to`: Rango de fechas para exportar posiciones.

- **Funcionalidad:** Exporta las posiciones a un archivo KML, que puede ser usado en Google Earth.
- **Permisos:** El método verifica si el usuario tiene permiso para acceder al dispositivo.

GET: Exportar a CSV (`getCsv`)

- **Parámetros:**
  - `deviceId`: El ID del dispositivo.
  - `from` y `to`: Rango de fechas para exportar posiciones.
- **Funcionalidad:** Exporta las posiciones a un archivo CSV, que puede ser abierto en software de hojas de cálculo.
- **Permisos:** Similar a la exportación KML, verifica los permisos del usuario.

GET: Exportar a GPX (`getGpx`)

- **Parámetros:**
  - `deviceId`: El ID del dispositivo.
  - `from` y `to`: Rango de fechas para exportar posiciones.
- **Funcionalidad:** Exporta las posiciones a un archivo GPX, que es comúnmente usado en dispositivos y aplicaciones GPS.
- **Permisos:** Verifica si el usuario tiene permiso para acceder al dispositivo.

Cómo Usar

- **Recuperar Posiciones:** Envía una solicitud `GET` al endpoint `/positions` con los parámetros adecuados (`deviceId`, `positionIds`, `from`, `to`) para recuperar las posiciones en formato JSON.
- **Eliminar Posiciones:** Envía una solicitud `DELETE` al endpoint `/positions` con `deviceId` y el rango de fechas `from` y `to` para eliminar posiciones específicas.
- **Exportar Posiciones:** Envía una solicitud `GET` a `/positions/kml`, `/positions/csv` o `/positions/gpx` con los parámetros `deviceId`, `from`, y `to` para exportar los datos en el formato deseado.

A continuación vamos a revisar cada una de estas dependencias para profundizar en el funcionamiento de la API:

II. BaseResource

A. *Descripción*

La clase BaseResource en la API de Traccar sirve como una clase base para todos los recursos de la API. Proporciona funcionalidades comunes como la obtención del ID de usuario autenticado y acceso a servicios compartidos como el almacenamiento y la gestión de permisos.

B. *Código*

```java
/*
 * 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;
    }

}
```

Listing 1. BASERESOURCE

C. *¿Cómo funciona?*

BaseResource facilita la interacción con los servicios de almacenamiento y permisos dentro de la API. También proporciona un método para obtener el ID del usuario actual autenticado a través del contexto de seguridad.

- **Acceso a servicios:** Los servicios de almacenamiento Storage y de permisos PermissionsService están inyectados en esta clase, permitiendo a las subclases interactuar con ellos fácilmente.
- **Obtención del ID de usuario (getUserId):** Este método extrae el ID del usuario autenticado del contexto de seguridad, lo cual es esencial para verificar permisos y realizar operaciones específicas del usuario.

D. *Funcionalidades Principales*

- **Inyección de dependencias:** `BaseResource` utiliza la inyección de dependencias para obtener instancias de `Storage` y `PermissionsService`.
- **Contexto de seguridad:** Maneja el contexto de seguridad para extraer información sobre el usuario autenticado.
- **Método utilitario para subclases:** Proporciona el método `getUserId` que es utilizado por las subclases para obtener el ID del usuario.

E. *Dependencias Principales*

- **org.traccar.api.security.PermissionsService**: Servicio que maneja la verificación de permisos para los usuarios.
- **org.traccar.api.security.UserPrincipal**: Representa al usuario autenticado en el contexto de seguridad.
- **org.traccar.storage.Storage**: Proporciona acceso a las operaciones de almacenamiento, como la recuperación y almacenamiento de datos.
- **jakarta.ws.rs.core.Context**: Anotación que se utiliza para inyectar el contexto de seguridad.
- **jakarta.ws.rs.core.SecurityContext**: Proporciona información sobre la autenticación y la seguridad de la solicitud actual.

III. PositionUtil

A. *Descripción*

La clase `PositionUtil` es una clase de utilidad en Traccar que proporciona métodos estáticos para trabajar con posiciones geográficas registradas por dispositivos. Esta clase incluye funciones para verificar si una posición es la más reciente, calcular distancias entre posiciones, y recuperar posiciones desde el almacenamiento.

IV. Código

```java
/*
 * 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.helper.model;

import org.traccar.model.BaseModel;
import org.traccar.model.Device;
import org.traccar.model.Position;
import org.traccar.model.User;
import org.traccar.session.cache.CacheManager;
import org.traccar.storage.Storage;
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 java.util.Date;
import java.util.List;
import java.util.stream.Collectors;

public final class PositionUtil {

    private PositionUtil() {
    }

    public static boolean isLatest(CacheManager cacheManager, Position position) {
        Position lastPosition = cacheManager.getPosition(position.getDeviceId());
        return lastPosition == null || position.getFixTime().compareTo(lastPosition.getFixTime()) >= 0;
    }

```

```
44      public static double calculateDistance(
            Position first, Position last, boolean
            useOdometer) {
45          double distance;
46          double firstOdometer = first.getDouble(
                Position.KEY_ODOMETER);
47          double lastOdometer = last.getDouble(
                Position.KEY_ODOMETER);
48
49          if (useOdometer && firstOdometer != 0.0 &&
                lastOdometer != 0.0) {
50              distance = lastOdometer - firstOdometer
                    ;
51          } else {
52              distance = last.getDouble(Position.
                    KEY_TOTAL_DISTANCE) - first.
                    getDouble(Position.
                    KEY_TOTAL_DISTANCE);
53          }
54          return distance;
55      }
56
57      public static List<Position> getPositions(
58              Storage storage, long deviceId, Date
                    from, Date to) throws
                    StorageException {
59          return storage.getObjects(Position.class,
                    new Request(
60                  new Columns.All(),
61                  new Condition.And(
62                          new Condition.Equals("deviceId
                                ", deviceId),
63                          new Condition.Between("fixTime
                                ", "from", from, "to", to)
                                ),
64                  new Order("fixTime")));
65      }
66
67      public static List<Position>
                getLatestPositions(Storage storage, long
                userId) throws StorageException {
68          var devices = storage.getObjects(Device.
                    class, new Request(
69                  new Columns.Include("id"),
70                  new Condition.Permission(User.class,
                            userId, Device.class)));
71          var deviceIds = devices.stream().map(
                    BaseModel::getId).collect(Collectors.
                    toUnmodifiableSet());
72
73          var positions = storage.getObjects(
                    Position.class, new Request(
74                  new Columns.All(), new Condition.
                        LatestPositions()));
75          return positions.stream()
76                  .filter(position -> deviceIds.
                        contains(position.getDeviceId())
                        )
77                  .collect(Collectors.toList());
78      }
79
80  }
```

Listing 2. PositionUtil

A. *¿Cómo funciona?*

`PositionUtil` actúa como una colección de métodos estáticos que simplifican las operaciones relacionadas con las posiciones en Traccar. Estos métodos no requieren una instancia de `PositionUtil` y son directamente accesibles.

- **Verificación de última posición (isLatest):** Comprueba si una posición dada es la más reciente para un dispositivo, comparando con la última posición almacenada en la caché.
- **Cálculo de distancia (calculateDistance):** Calcula la distancia entre dos posiciones usando el odómetro o la distancia total registrada, según la disponibilidad de los datos.
- **Recuperación de posiciones (getPositions):** Obtiene una lista de posiciones para un dispositivo específico dentro de un rango de fechas desde el almacenamiento.
- **Obtención de últimas posiciones (getLatestPositions):** Recupera las últimas posiciones de todos los dispositivos asociados a un usuario desde el almacenamiento.

B. *Funcionalidades Principales*

- **Detección de última posición:** Verifica si una posición es la más reciente registrada para un dispositivo.
- **Cálculo de distancias:** Calcula la distancia entre dos posiciones usando diferentes métodos según la disponibilidad de datos.
- **Consultas a la base de datos:** Proporciona métodos para consultar posiciones de la base de datos dentro de un rango de tiempo o para obtener las últimas posiciones de dispositivos específicos.

C. *Dependencias Principales*

- **org.traccar.model.BaseModel**: Clase base para los modelos en Traccar, proporcionando identificadores únicos y funcionalidades comunes.
- **org.traccar.model.Device**: Representa un dispositivo de rastreo en Traccar.
- **org.traccar.model.Position**: Representa una posición geográfica registrada por un dispositivo.

- **org.traccar.model.User**: Representa un usuario del sistema Traccar.
- **org.traccar.session.cache.CacheManager**: Gestiona la caché de sesiones y posiciones en Traccar.
- **org.traccar.storage.Storage**: Proporciona acceso a las operaciones de almacenamiento, como la recuperación y almacenamiento de datos.
- **org.traccar.storage.StorageException**: Excepción lanzada en caso de problemas con la base de datos o almacenamiento.
- **org.traccar.storage.query.Columns**: Utilizado para especificar las columnas a recuperar en una consulta.
- **org.traccar.storage.query.Condition**: Utilizado para definir condiciones en las consultas a la base de datos.
- **org.traccar.storage.query.Order**: Define el orden de los resultados en una consulta a la base de datos.
- **org.traccar.storage.query.Request**: Clase que encapsula una solicitud de consulta a la base de datos.

V. Device

A. *Descripción*

La clase `Device` representa un dispositivo de rastreo en el sistema Traccar. Esta clase almacena información relevante sobre el dispositivo, como su identificador único, estado, modelo, y otros atributos relacionados con su funcionamiento y administración.

B. *¿Cómo funciona?*

La clase `Device` es utilizada para modelar los dispositivos rastreados en Traccar. Incluye atributos que permiten la identificación del dispositivo, su estado actual, y configuraciones específicas como la asignación de calendarios y la habilitación o deshabilitación del dispositivo. También proporciona métodos para manipular estos atributos y obtener su información.

VI. Código

```java
/*
 * Copyright 2012 - 2023 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 com.fasterxml.jackson.annotation.JsonIgnore;
import org.traccar.storage.QueryIgnore;
import org.traccar.storage.StorageName;

import java.util.Date;

@StorageName("tc_devices")
public class Device extends GroupedModel implements Disableable, Schedulable {

    private long calendarId;

    @Override
    public long getCalendarId() {
        return calendarId;
    }

    @Override
    public void setCalendarId(long calendarId) {
        this.calendarId = calendarId;
    }

    private String name;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    private String uniqueId;

    public String getUniqueId() {
        return uniqueId;
    }

    public void setUniqueId(String uniqueId) {
        if (uniqueId.contains("..")) {
            throw new IllegalArgumentException("Invalid unique id");
        }
        this.uniqueId = uniqueId.trim();
    }

    public static final String STATUS_UNKNOWN = "unknown";
    public static final String STATUS_ONLINE = "online";
    public static final String STATUS_OFFLINE = "offline";
```

```
65
66      private String status;
67
68      @QueryIgnore
69      public String getStatus() {
70          return status != null ? status :
                STATUS_OFFLINE;
71      }
72
73      public void setStatus(String status) {
74          this.status = status != null ? status.trim
                () : null;
75      }
76
77      private Date lastUpdate;
78
79      @QueryIgnore
80      public Date getLastUpdate() {
81          return this.lastUpdate;
82      }
83
84      public void setLastUpdate(Date lastUpdate) {
85          this.lastUpdate = lastUpdate;
86      }
87
88      private long positionId;
89
90      @QueryIgnore
91      public long getPositionId() {
92          return positionId;
93      }
94
95      public void setPositionId(long positionId) {
96          this.positionId = positionId;
97      }
98
99      private String phone;
100
101     public String getPhone() {
102         return phone;
103     }
104
105     public void setPhone(String phone) {
106         this.phone = phone != null ? phone.trim()
                : null;
107     }
108
109     private String model;
110
111     public String getModel() {
112         return model;
113     }
114
115     public void setModel(String model) {
116         this.model = model;
117     }
118
119     private String contact;
120
121     public String getContact() {
122         return contact;
123     }
124
125     public void setContact(String contact) {
126         this.contact = contact;
127     }
128
129     private String category;
130
131     public String getCategory() {
132         return category;
133     }
134
135     public void setCategory(String category) {
136         this.category = category;
137     }
138
139     private boolean disabled;
140
141     @Override
142     public boolean getDisabled() {
143         return disabled;
144     }
145
146     @Override
147     public void setDisabled(boolean disabled) {
148         this.disabled = disabled;
149     }
150
151     private Date expirationTime;
152
153     @Override
154     public Date getExpirationTime() {
155         return expirationTime;
156     }
157
158     @Override
159     public void setExpirationTime(Date
            expirationTime) {
160         this.expirationTime = expirationTime;
161     }
162
163     private boolean motionStreak;
164
165     @QueryIgnore
166     @JsonIgnore
167     public boolean getMotionStreak() {
168         return motionStreak;
169     }
170
171     @JsonIgnore
172     public void setMotionStreak(boolean
            motionStreak) {
173         this.motionStreak = motionStreak;
174     }
175
176     private boolean motionState;
177
178     @QueryIgnore
179     @JsonIgnore
180     public boolean getMotionState() {
181         return motionState;
182     }
183
184     @JsonIgnore
185     public void setMotionState(boolean
            motionState) {
186         this.motionState = motionState;
187     }
188
189     private Date motionTime;
190
191     @QueryIgnore
192     @JsonIgnore
193     public Date getMotionTime() {
194         return motionTime;
195     }
196
197     @JsonIgnore
198     public void setMotionTime(Date motionTime) {
199         this.motionTime = motionTime;
200     }
201
202     private double motionDistance;
203
204     @QueryIgnore
```

```
205     @JsonIgnore
206     public double getMotionDistance() {
207        return motionDistance;
208     }
209
210     @JsonIgnore
211     public void setMotionDistance(double
            motionDistance) {
212        this.motionDistance = motionDistance;
213     }
214
215     private boolean overspeedState;
216
217     @QueryIgnore
218     @JsonIgnore
219     public boolean getOverspeedState() {
220        return overspeedState;
221     }
222
223     @JsonIgnore
224     public void setOverspeedState(boolean
            overspeedState) {
225        this.overspeedState = overspeedState;
226     }
227
228     private Date overspeedTime;
229
230     @QueryIgnore
231     @JsonIgnore
232     public Date getOverspeedTime() {
233        return overspeedTime;
234     }
235
236     @JsonIgnore
237     public void setOverspeedTime(Date
            overspeedTime) {
238        this.overspeedTime = overspeedTime;
239     }
240
241     private long overspeedGeofenceId;
242
243     @QueryIgnore
244     @JsonIgnore
245     public long getOverspeedGeofenceId() {
246        return overspeedGeofenceId;
247     }
248
249     @JsonIgnore
250     public void setOverspeedGeofenceId(long
            overspeedGeofenceId) {
251        this.overspeedGeofenceId =
               overspeedGeofenceId;
252     }
253
254 }
```

Listing 3. Device

- **Identificación del dispositivo:** Atributos como `uniqueId`, `name`, y `phone` permiten identificar de manera única al dispositivo en el sistema.

- **Estado del dispositivo:** El atributo `status` indica si el dispositivo está `online`, `offline` o en un estado `unknown`. Otros atributos como `motionState` y `overspeedState` rastrean el movimiento y exceso de velocidad, respectivamente.

- **Configuraciones de administración:** Atributos como `disabled`, `expirationTime`, y `calendarId` permiten la gestión del dispositivo, incluyendo su habilitación/deshabilitación y la programación de su actividad.

A. *Funcionalidades Principales*

- **Identificación única:** Cada dispositivo tiene un `uniqueId` que lo distingue de otros dispositivos en el sistema.

- **Gestión del estado:** Permite rastrear el estado actual del dispositivo y su última actualización.

- **Control de movimiento:** Registra y permite consultar el estado de movimiento, incluyendo la detección de exceso de velocidad.

- **Programación y deshabilitación:** La clase permite configurar la programación y deshabilitar temporalmente el dispositivo según sea necesario.

B. *Dependencias Principales*

- **com.fasterxml.jackson.annotation.JsonIgnore**: Anotación utilizada para ignorar campos durante la serialización/deserialización JSON.

- **org.traccar.storage.QueryIgnore**: Anotación que indica que un campo debe ser ignorado en consultas a la base de datos.

- **org.traccar.storage.StorageName**: Anotación que define el nombre de la tabla de base de datos donde se almacena la clase.
- **org.traccar.model.GroupedModel**: Clase base que proporciona la funcionalidad de agrupación para modelos en Traccar.
- **org.traccar.model.Disableable**: Interfaz que permite habilitar o deshabilitar un dispositivo.

- **org.traccar.model.Schedulable**: Interfaz que permite manejar la programación de dispositivos, como la asignación de calendarios.

VII. Position-Model

A. *Descripción*

La clase `Position` en Traccar es responsable de almacenar y manejar la información de la posición de un dispositivo de rastreo GPS. Extiende la clase `Message` y contiene múltiples atributos que describen detalles específicos sobre la ubicación y el estado del dispositivo en un momento dado.

VIII. Código

```java
/*
 * Copyright 2012 - 2024 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 java.util.Date;
import java.util.List;
import java.util.stream.Collectors;

import com.fasterxml.jackson.annotation.JsonIgnore;
import org.traccar.storage.QueryIgnore;
import org.traccar.storage.StorageName;

@StorageName("tc_positions")
public class Position extends Message {

    public static final String KEY_ORIGINAL = "raw";
    public static final String KEY_INDEX = "index";
    public static final String KEY_HDOP = "hdop";
    public static final String KEY_VDOP = "vdop";
    public static final String KEY_PDOP = "pdop";
    public static final String KEY_SATELLITES = "sat"; // in use
    public static final String KEY_SATELLITES_VISIBLE = "satVisible";
    public static final String KEY_RSSI = "rssi";
    public static final String KEY_GPS = "gps";
    public static final String KEY_ROAMING = "roaming";
    public static final String KEY_EVENT = "event";
    public static final String KEY_ALARM = "alarm";
    public static final String KEY_STATUS = "status";
    public static final String KEY_ODOMETER = "odometer"; // meters
    public static final String KEY_ODOMETER_SERVICE = "serviceOdometer"; // meters
    public static final String KEY_ODOMETER_TRIP = "tripOdometer"; // meters
    public static final String KEY_HOURS = "hours"; // milliseconds
    public static final String KEY_STEPS = "steps";
    public static final String KEY_HEART_RATE = "heartRate";
    public static final String KEY_INPUT = "input";
    public static final String KEY_OUTPUT = "output";
    public static final String KEY_IMAGE = "image";
    public static final String KEY_VIDEO = "video";
    public static final String KEY_AUDIO = "audio";

    // The units for the below four KEYs currently vary.
    // The preferred units of measure are specified in the comment for each.
    public static final String KEY_POWER = "power"; // volts
    public static final String KEY_BATTERY = "battery"; // volts
    public static final String KEY_BATTERY_LEVEL = "batteryLevel"; // percentage
    public static final String KEY_FUEL_LEVEL = "fuel"; // liters
    public static final String KEY_FUEL_USED = "fuelUsed"; // liters
    public static final String KEY_FUEL_CONSUMPTION = "fuelConsumption"; // liters/hour

    public static final String KEY_VERSION_FW = "versionFw";
    public static final String KEY_VERSION_HW = "versionHw";
    public static final String KEY_TYPE = "type";
    public static final String KEY_IGNITION = "ignition";
    public static final String KEY_FLAGS = "flags";
    public static final String KEY_ANTENNA = "antenna";
    public static final String KEY_CHARGE = "charge";
    public static final String KEY_IP = "ip";
    public static final String KEY_ARCHIVE = "archive";
    public static final String KEY_DISTANCE = "distance"; // meters
    public static final String KEY_TOTAL_DISTANCE = "totalDistance"; // meters
    public static final String KEY_RPM = "rpm";
    public static final String KEY_VIN = "vin";
    public static final String KEY_APPROXIMATE = "approximate";
    public static final String KEY_THROTTLE = "throttle";
    public static final String KEY_MOTION = "motion";
    public static final String KEY_ARMED = "armed";
```

```
    public static final String KEY_GEOFENCE = "geofence";
    public static final String KEY_ACCELERATION = "acceleration";
    public static final String KEY_DEVICE_TEMP = "deviceTemp"; // celsius
    public static final String KEY_COOLANT_TEMP = "coolantTemp"; // celsius
    public static final String KEY_ENGINE_LOAD = "engineLoad";
    public static final String KEY_OPERATOR = "operator";
    public static final String KEY_COMMAND = "command";
    public static final String KEY_BLOCKED = "blocked";
    public static final String KEY_LOCK = "lock";
    public static final String KEY_DOOR = "door";
    public static final String KEY_AXLE_WEIGHT = "axleWeight";
    public static final String KEY_G_SENSOR = "gSensor";
    public static final String KEY_ICCID = "iccid";
    public static final String KEY_PHONE = "phone";
    public static final String KEY_SPEED_LIMIT = "speedLimit";
    public static final String KEY_DRIVING_TIME = "drivingTime";

    public static final String KEY_DTCS = "dtcs";
    public static final String KEY_OBD_SPEED = "obdSpeed"; // km/h
    public static final String KEY_OBD_ODOMETER = "obdOdometer"; // meters

    public static final String KEY_RESULT = "result";

    public static final String KEY_DRIVER_UNIQUE_ID = "driverUniqueId";
    public static final String KEY_CARD = "card";

    // Start with 1 not 0
    public static final String PREFIX_TEMP = "temp";
    public static final String PREFIX_ADC = "adc";
    public static final String PREFIX_IO = "io";
    public static final String PREFIX_COUNT = "count";
    public static final String PREFIX_IN = "in";
    public static final String PREFIX_OUT = "out";

    public static final String ALARM_GENERAL = "general";
    public static final String ALARM_SOS = "sos";
    public static final String ALARM_VIBRATION = "vibration";
    public static final String ALARM_MOVEMENT = "movement";
    public static final String ALARM_LOW_SPEED = "lowspeed";
    public static final String ALARM_OVERSPEED = "overspeed";
    public static final String ALARM_FALL_DOWN = "fallDown";
    public static final String ALARM_LOW_POWER = "lowPower";
    public static final String ALARM_LOW_BATTERY = "lowBattery";
    public static final String ALARM_FAULT = "fault";
    public static final String ALARM_POWER_OFF = "powerOff";
    public static final String ALARM_POWER_ON = "powerOn";
    public static final String ALARM_DOOR = "door";
    public static final String ALARM_LOCK = "lock";
    public static final String ALARM_UNLOCK = "unlock";
    public static final String ALARM_GEOFENCE = "geofence";
    public static final String ALARM_GEOFENCE_ENTER = "geofenceEnter";
    public static final String ALARM_GEOFENCE_EXIT = "geofenceExit";
    public static final String ALARM_GPS_ANTENNA_CUT = "gpsAntennaCut";
    public static final String ALARM_ACCIDENT = "accident";
    public static final String ALARM_TOW = "tow";
    public static final String ALARM_IDLE = "idle";
    public static final String ALARM_HIGH_RPM = "highRpm";
    public static final String ALARM_ACCELERATION = "hardAcceleration";
    public static final String ALARM_BRAKING = "hardBraking";
    public static final String ALARM_CORNERING = "hardCornering";
    public static final String ALARM_LANE_CHANGE = "laneChange";
    public static final String ALARM_FATIGUE_DRIVING = "fatigueDriving";
    public static final String ALARM_POWER_CUT = "powerCut";
    public static final String ALARM_POWER_RESTORED = "powerRestored";
    public static final String ALARM_JAMMING = "jamming";
    public static final String ALARM_TEMPERATURE = "temperature";
    public static final String ALARM_PARKING = "parking";
    public static final String ALARM_BONNET = "bonnet";
    public static final String ALARM_FOOT_BRAKE = "footBrake";
    public static final String ALARM_FUEL_LEAK = "fuelLeak";
    public static final String ALARM_TAMPERING = "tampering";
    public static final String ALARM_REMOVING = "removing";

    public Position() {
    }

    public Position(String protocol) {
        this.protocol = protocol;
    }

    private String protocol;

    public String getProtocol() {
        return protocol;
    }

    public void setProtocol(String protocol) {
        this.protocol = protocol;
```

```
    }

    private Date serverTime = new Date();

    public Date getServerTime() {
        return serverTime;
    }

    public void setServerTime(Date serverTime) {
        this.serverTime = serverTime;
    }

    private Date deviceTime;

    public Date getDeviceTime() {
        return deviceTime;
    }

    public void setDeviceTime(Date deviceTime) {
        this.deviceTime = deviceTime;
    }

    private Date fixTime;

    public Date getFixTime() {
        return fixTime;
    }

    public void setFixTime(Date fixTime) {
        this.fixTime = fixTime;
    }

    @QueryIgnore
    public void setTime(Date time) {
        setDeviceTime(time);
        setFixTime(time);
    }

    private boolean outdated;

    @QueryIgnore
    public boolean getOutdated() {
        return outdated;
    }

    @QueryIgnore
    public void setOutdated(boolean outdated) {
        this.outdated = outdated;
    }

    private boolean valid;

    public boolean getValid() {
        return valid;
    }

    public void setValid(boolean valid) {
        this.valid = valid;
    }

    private double latitude;

    public double getLatitude() {
        return latitude;
    }

    public void setLatitude(double latitude) {
        if (latitude < -90 || latitude > 90) {
            throw new IllegalArgumentException("
                Latitude out of range");
        }
        this.latitude = latitude;
    }

    private double longitude;

    public double getLongitude() {
        return longitude;
    }

    public void setLongitude(double longitude) {
        if (longitude < -180 || longitude > 180) {
            throw new IllegalArgumentException("
                Longitude out of range");
        }
        this.longitude = longitude;
    }

    private double altitude; // value in meters

    public double getAltitude() {
        return altitude;
    }

    public void setAltitude(double altitude) {
        this.altitude = altitude;
    }

    private double speed; // value in knots

    public double getSpeed() {
        return speed;
    }

    public void setSpeed(double speed) {
        this.speed = speed;
    }

    private double course;

    public double getCourse() {
        return course;
    }

    public void setCourse(double course) {
        this.course = course;
    }

    private String address;

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    private double accuracy;

    public double getAccuracy() {
        return accuracy;
    }

    public void setAccuracy(double accuracy) {
        this.accuracy = accuracy;
    }

    private Network network;

    public Network getNetwork() {
        return network;
    }

    public void setNetwork(Network network) {
        this.network = network;
```

```
    }

    private List<Long> geofenceIds;

    public List<Long> getGeofenceIds() {
        return geofenceIds;
    }

    public void setGeofenceIds(List<? extends
         Number> geofenceIds) {
        if (geofenceIds != null) {
            this.geofenceIds = geofenceIds.stream()
                 .map(Number::longValue).collect(
                 Collectors.toList());
        } else {
            this.geofenceIds = null;
        }
    }

    public void addAlarm(String alarm) {
        if (alarm != null) {
            if (hasAttribute(KEY_ALARM)) {
                set(KEY_ALARM, getAttributes().get(
                     KEY_ALARM) + "," + alarm);
            } else {
                set(KEY_ALARM, alarm);
            }
        }
    }

    @JsonIgnore
    @QueryIgnore
    @Override
    public String getType() {
        return super.getType();
    }

    @JsonIgnore
    @QueryIgnore
    @Override
    public void setType(String type) {
        super.setType(type);
    }

}
```

Listing 4. Position

A. *¿Cómo funciona?*

La clase `Position` encapsula los datos recibidos de los dispositivos de rastreo y los almacena en la base de datos. Estos datos incluyen información de la posición, como las coordenadas GPS, la velocidad, la dirección, entre otros. Además, registra eventos específicos como alarmas o la detección de movimiento.

- **Información de la posición:** La clase almacena atributos esenciales como latitud, longitud, velocidad, altitud y rumbo, que permiten determinar la ubicación y movimiento del dispositivo.
- **Eventos del dispositivo:** Atributos como `alarm`, `event`, y `status` permiten registrar y manejar eventos relevantes del dispositivo, como la activación de una alarma o un cambio en su estado.
- **Datos adicionales:** Además de la ubicación, la clase puede contener información adicional como el nivel de batería, la calidad de la señal GPS, y los datos específicos del dispositivo.

B. *Funcionalidades Principales*

- **Almacenamiento de coordenadas GPS:** Registra y almacena la latitud y longitud del dispositivo, permitiendo su rastreo en tiempo real.
- **Monitoreo de velocidad y dirección:** Permite registrar la velocidad y la dirección del dispositivo en movimiento.
- **Registro de eventos:** Mantiene un registro de eventos importantes, como alarmas o cambios de estado.
- **Soporte para datos personalizados:** Los atributos adicionales permiten almacenar datos específicos del dispositivo, como el nivel de batería o la cantidad de satélites en uso.

C. *Dependencias Principales*

- **org.traccar.storage.StorageName**: Anotación que define la tabla de base de datos (`tc_positions`) donde se almacenan las instancias de `Position`.
- **org.traccar.model.Message**: Clase base de la que `Position` hereda, proporcionando funcionalidades comunes para manejar mensajes del dispositivo.

IX. UserRestrictions

A. *Descripción*

La interfaz `UserRestrictions` define las restricciones que pueden aplicarse a un usuario en el sistema Traccar. Proporciona métodos para consultar diversas restricciones que pueden afectar el acceso y la capacidad del usuario para interactuar con el sistema, como la lectura de datos, el uso de comandos, la generación de reportes, y la configuración de correos electrónicos.

X. CÓDIGO

```java
/*
 * 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.model;

public interface UserRestrictions {
    boolean getReadonly();
    boolean getDeviceReadonly();
    boolean getLimitCommands();
    boolean getDisableReports();
    boolean getFixedEmail();
}
```

Listing 5. UserRestrictions

A. *¿Cómo funciona?*

La interfaz `UserRestrictions` establece un contrato para las clases que implementan restricciones de usuario. Los métodos definidos en esta interfaz permiten determinar si ciertas restricciones están habilitadas para un usuario específico. Las implementaciones concretas de esta interfaz serán responsables de proporcionar la lógica que define cómo se aplican estas restricciones.

- **Acceso en solo lectura:** El método `getReadonly()` indica si el usuario tiene acceso en solo lectura, lo que limita la capacidad de modificar datos en el sistema.
- **Acceso en solo lectura a dispositivos:** El método `getDeviceReadonly()` determina si el usuario tiene permisos de solo lectura específicamente para los dispositivos.
- **Limitación de comandos:** El método `getLimitCommands()` indica si el usuario tiene restricciones en la ejecución de comandos en el sistema.
- **Desactivación de reportes:** El método `getDisableReports()` determina si la capacidad de generar reportes está desactivada para el usuario.
- **Correo electrónico fijo:** El método `getFixedEmail()` indica si el correo electrónico del usuario está fijo, lo que puede limitar la capacidad de cambiar la dirección de correo electrónico asociada a la cuenta.

B. *Funcionalidades Principales*

- **Consulta de restricciones de lectura:** Permite verificar si el usuario tiene restricciones para leer o modificar datos en el sistema.
- **Verificación de permisos de comando:** Determina si el usuario puede ejecutar comandos y qué comandos están limitados.
- **Control de generación de reportes:** Controla la capacidad del usuario para generar reportes dentro del sistema.
- **Gestión de correos electrónicos:** Permite verificar si el usuario puede cambiar su dirección de correo electrónico o si esta está fija.

C. *Dependencias Principales*

- **No tiene dependencias externas específicas:** La interfaz `UserRestrictions` se define como una interfaz independiente sin dependencias externas específicas a otras clases o bibliotecas.

XI. CSVEXPORTPROVIDER

A. *Descripción*

La clase `CsvExportProvider` es responsable de exportar datos de posiciones en formato CSV. Utiliza el almacenamiento para recuperar los datos de posiciones y luego genera un archivo CSV con la información requerida. Esta clase es parte del paquete `org.traccar.reports` y proporciona una funcionalidad clave para la exportación de datos en el sistema Traccar.

XII. Código

```java
/*
 * 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.reports;

import org.traccar.helper.DateUtil;
import org.traccar.helper.model.PositionUtil;
import org.traccar.model.Position;
import org.traccar.storage.Storage;
import org.traccar.storage.StorageException;

import jakarta.inject.Inject;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.util.Date;
import java.util.LinkedHashMap;
import java.util.Objects;
import java.util.function.Function;
import java.util.stream.Collectors;

public class CsvExportProvider {

    private final Storage storage;

    @Inject
    public CsvExportProvider(Storage storage) {
        this.storage = storage;
    }

    public void generate(
            OutputStream outputStream, long deviceId, Date from, Date to)
            throws StorageException {

        var positions = PositionUtil.getPositions(storage, deviceId, from, to);

        var attributes = positions.stream()
                .flatMap((position -> position.getAttributes().keySet().stream()))
                .collect(Collectors.toUnmodifiableSet());

        var properties = new LinkedHashMap<String, Function<Position, Object>>();
        properties.put("id", Position::getId);
        properties.put("deviceId", Position::getDeviceId);
        properties.put("protocol", Position::getProtocol);
        properties.put("serverTime", position -> DateUtil.formatDate(position.getServerTime()));
        properties.put("deviceTime", position -> DateUtil.formatDate(position.getDeviceTime()));
        properties.put("fixTime", position -> DateUtil.formatDate(position.getFixTime()));
        properties.put("valid", Position::getValid);
        properties.put("latitude", Position::getLatitude);
        properties.put("longitude", Position::getLongitude);
        properties.put("altitude", Position::getAltitude);
        properties.put("speed", Position::getSpeed);
        properties.put("course", Position::getCourse);
        properties.put("address", Position::getAddress);
        properties.put("accuracy", Position::getAccuracy);
        attributes.forEach(key -> properties.put(key, position -> position.getAttributes().get(key)));

        try (PrintWriter writer = new PrintWriter(outputStream)) {
            writer.println(String.join(",", properties.keySet()));
            positions.forEach(position -> writer.println(properties.values().stream()
                    .map(f -> Objects.toString(f.apply(position), ""))
                    .collect(Collectors.joining(","))));
        }
    }

}
```

Listing 6. CsvExportProvider

A. *¿Cómo funciona?*

La clase `CsvExportProvider` obtiene los datos de posiciones de un dispositivo en un rango de fechas especificado y los exporta en formato CSV. Los datos se recuperan mediante el uso de la clase `PositionUtil` y se procesan para incluir tanto los atributos estándar como los específicos de cada posición. Los datos exportados incluyen identificadores, tiempos, coordenadas y atributos adicionales asociados con cada posición.

- **Recuperación de posiciones:** Utiliza el método `getPositions` de `PositionUtil` para recuperar las posiciones del dispositivo dentro del rango de fechas dado.

- **Procesamiento de atributos:** Extrae y procesa todos los atributos de las posiciones para incluirlos en el archivo CSV.

- **Generación del archivo CSV:** Escribe los datos de las posiciones en un archivo CSV utilizando un `PrintWriter` para generar el contenido del archivo.

B. *Funcionalidades Principales*

- **Exportación de datos en CSV:** Permite exportar datos de posiciones en un formato CSV legible y estructurado.

- **Inclusión de atributos personalizados:** Agrega atributos personalizados a la exportación, permitiendo la flexibilidad en el formato de los datos exportados.

- **Configuración dinámica de campos:** Permite la configuración dinámica de campos a incluir en el archivo CSV, basándose en los atributos de las posiciones.

C. *Dependencias Principales*

- **org.traccar.helper.DateUtil**: Utilidad para el formato de fechas.

- **org.traccar.helper.model.PositionUtil**: Utilidad para la obtención de posiciones del almacenamiento.

- **org.traccar.model.Position**: Representa la entidad de posición que se exporta.

- **org.traccar.storage.Storage**: Interfaz para el almacenamiento de datos.

- **org.traccar.storage.StorageException**: Excepción lanzada en caso de errores en el almacenamiento.

XIII. KmlExportProvider

A. *Descripción*

La clase `KmlExportProvider` se encarga de exportar datos de posiciones en formato KML (Keyhole Markup Language). Utiliza el almacenamiento para recuperar los datos de posiciones de un dispositivo y los exporta en un archivo KML. Este formato es útil para visualizar datos de rastreo en aplicaciones que soportan KML, como Google Earth.

XIV. Código

```
/*
 * 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.reports;

import org.traccar.helper.DateUtil;
import org.traccar.helper.model.PositionUtil;
import org.traccar.model.Device;
import org.traccar.storage.Storage;
import org.traccar.storage.StorageException;
import org.traccar.storage.query.Columns;
import org.traccar.storage.query.Condition;
import org.traccar.storage.query.Request;

import jakarta.inject.Inject;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.util.Date;

public class GpxExportProvider {

    private final Storage storage;

    @Inject
    public GpxExportProvider(Storage storage) {
        this.storage = storage;
    }

    public void generate(
            OutputStream outputStream, long deviceId, Date from, Date to)
            throws StorageException {

        var device = storage.getObject(Device.class, new Request(
                new Columns.All(), new Condition.Equals("id", deviceId)));
        var positions = PositionUtil.getPositions(storage, deviceId, from, to);

        try (PrintWriter writer = new PrintWriter(outputStream)) {
            writer.print("<?xml version=\"1.0\" encoding=\"UTF-8\"?>");
            writer.print("<gpx version=\"1.0\">");
            writer.print("<trk>");
            writer.print("<name>");
```

```
53            writer.print(device.getName());
54            writer.print("</name>");
55            writer.print("<trkseg>");
56            positions.forEach(position -> {
57                writer.print("<trkpt lat=\"");
58                writer.print(position.getLatitude()
                      );
59                writer.print("\" lon=\"");
60                writer.print(position.getLongitude()
                      );
61                writer.print("\">");
62                writer.print("<ele>");
63                writer.print(position.getAltitude()
                      );
64                writer.print("</ele>");
65                writer.print("<time>");
66                writer.print(DateUtil.formatDate(
                      position.getFixTime()));
67                writer.print("</time>");
68                writer.print("</trkpt>");
69            });
70            writer.print("</trkseg>");
71            writer.print("</trk>");
72            writer.print("</gpx>");
73        }
74    }
75
76 }
```

Listing 7. GpxExportProvider

A. *¿Cómo funciona?*

La clase `KmlExportProvider` obtiene los datos de un dispositivo y las posiciones en un rango de fechas especificado y los exporta en formato KML. Primero, recupera la información del dispositivo utilizando el almacenamiento y luego genera un archivo KML que incluye una línea representando el recorrido del dispositivo con atributos como longitud, latitud y altitud.

- **Recuperación de dispositivo:** Utiliza el almacenamiento para obtener el dispositivo basado en su identificador.
- **Recuperación de posiciones:** Obtiene las posiciones del dispositivo en el rango de fechas dado utilizando la utilidad `PositionUtil`.
- **Generación del archivo KML:** Escribe los datos en formato KML, incluyendo el nombre del dispositivo y una línea que representa el recorrido del dispositivo con atributos como longitud, latitud, altitud y rango de fechas.

B. *Funcionalidades Principales*

- **Exportación en formato KML:** Permite exportar datos de posiciones en un formato KML estándar.
- **Inclusión de información del dispositivo:** Incluye el nombre del dispositivo en el archivo KML.
- **Generación de línea de recorrido:** Representa el recorrido del dispositivo como una línea en el archivo KML, incluyendo longitud, latitud y altitud.

C. *Dependencias Principales*

- **org.traccar.helper.model.PositionUtil**: Utilidad para la obtención de posiciones del almacenamiento.
- **org.traccar.model.Device**: Representa la entidad del dispositivo cuyos datos se incluyen en el archivo KML.
- **org.traccar.storage.Storage**: Interfaz para el almacenamiento de datos.
- **org.traccar.storage.StorageException**: Excepción lanzada en caso de errores en el almacenamiento.
- **org.traccar.storage.query.Columns**: Clase utilizada para definir las columnas de consulta.
- **org.traccar.storage.query.Condition**: Clase utilizada para definir las condiciones de consulta.
- **org.traccar.storage.query.Request**: Clase utilizada para crear solicitudes de consulta.

XV. StorageException

A. *Descripción*

La clase `StorageException` es una excepción personalizada utilizada en el sistema Traccar para manejar errores relacionados con el almacenamiento de datos. Hereda de `Exception` y proporciona varias formas de inicializar la excepción con mensajes y causas.

XVI. Código

```
1 /*
2  * Copyright 2022 Anton Tananaev (anton@traccar.
      org)
3  *
```

```
 * 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);
    }

}
```

Listing 8. StorageException

A. *¿Cómo funciona?*

La clase `StorageException` extiende `Exception` para ofrecer un manejo específico de errores en el contexto de operaciones de almacenamiento. Permite crear excepciones con un mensaje descriptivo, una causa específica, o ambos. Esta clase es útil para capturar y gestionar errores que ocurren durante la interacción con el almacenamiento de datos en Traccar.

- **Inicialización con mensaje:** Permite inicializar la excepción con un mensaje que describe el error ocurrido.
- **Inicialización con causa:** Permite inicializar la excepción con una causa subyacente que representa el error original.
- **Inicialización con mensaje y causa:** Permite proporcionar tanto un mensaje descriptivo como una causa que detalla el error.

B. *Funcionalidades Principales*

- **Constructor con mensaje:** `StorageException(String message)` permite crear una excepción con un mensaje que describe el problema.
- **Constructor con causa:** `StorageException(Throwable cause)` permite crear una excepción que envuelve una causa original.
- **Constructor con mensaje y causa:** `StorageException(String message, Throwable cause)` permite crear una excepción con ambos, mensaje y causa.

C. *Dependencias Principales*

- **java.lang.Exception**: Clase base de la cual `StorageException` hereda.

XVII. Columns

A. *Descripción*

La clase `Columns` proporciona una estructura abstracta para definir y gestionar columnas en consultas de base de datos. Permite obtener listas de nombres de columnas basadas en los métodos de una clase y en tipos específicos de métodos (por ejemplo, getters o setters). Incluye subclases para incluir o excluir columnas según las necesidades.

XVIII. Código

```
/*
 * 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;

```

```java
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());
        }
    }

}
```

Listing 9. Columns

A. *¿Cómo funciona?*

La clase `Columns` define un método abstracto `getColumns` que debe ser implementado por sus subclases para obtener columnas específicas basadas en una clase y un tipo de método. Proporciona implementaciones para obtener todas las columnas, incluir columnas específicas, o excluir columnas específicas.

- **Obtención de columnas:** La clase base permite obtener nombres de columnas a partir de métodos de una clase, excluyendo métodos anotados con `@QueryIgnore` y el método `getClass()`.
- **Subclases especializadas:** `All`, `Include`, y `Exclude` ofrecen diferentes estrategias para obtener listas de columnas.

B. *Funcionalidades Principales*

- **Obtener todas las columnas:** La subclase `All` obtiene todos los nombres de columnas posibles de una clase según los métodos de tipo getter o setter.
- **Incluir columnas específicas:** La subclase `Include` permite especificar un conjunto de columnas que deben ser incluidas en la consulta.
- **Excluir columnas específicas:** La subclase `Exclude` permite excluir un conjunto de columnas de la lista de columnas obtenidas.

C. *Dependencias Principales*

- **org.traccar.storage.QueryIgnore**: Anotación utilizada para ignorar métodos durante la obtención de columnas.
- **java.beans.Introspector**: Utilizado para decapitar nombres de métodos y obtener nombres de columnas.

- **java.lang.reflect.Method**: Utilizado para obtener métodos de una clase y determinar si deben ser incluidos como columnas.

XIX. Condition

A. *Descripción*

La interfaz `Condition` define una estructura para representar condiciones en consultas de base de datos en el sistema Traccar. Permite la creación de condiciones simples y compuestas, así como la combinación de múltiples condiciones usando operadores lógicos.

XX. Código

```
/*
 * 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;
    }
}

}
```

Listing 10. Condition

A. *¿Cómo funciona?*

La interfaz `Condition` proporciona una forma de definir condiciones para consultas, con la capacidad de combinar estas condiciones utilizando operadores lógicos como `AND` y `OR`. Incluye implementaciones específicas para diferentes tipos de condiciones, como igualdad y rango.

- **Condiciones básicas:** Las subclases como `Equals` y `Between` permiten definir condiciones de igualdad y de rango de valores, respectivamente.
- **Composición de condiciones:** Las subclases `And` y `Or` permiten combinar condiciones usando operadores lógicos.

- **Permisos:** La subclase `Permission` define condiciones relacionadas con permisos de acceso a recursos basados en el propietario y la propiedad.
- **Últimas posiciones:** La subclase `LatestPositions` permite obtener las últimas posiciones de un dispositivo específico.

B. *Funcionalidades Principales*

- **Definición de igualdad:** La subclase `Equals` permite definir una condición de igualdad para una columna específica.
- **Definición de rango:** La subclase `Between` permite definir una condición para un rango de valores.
- **Composición lógica:** Las subclases `And` y `Or` permiten combinar múltiples condiciones usando operadores lógicos.
- **Permisos de acceso:** La subclase `Permission` permite definir condiciones relacionadas con permisos y exclusión de grupos.
- **Últimas posiciones de dispositivos:** La subclase `LatestPositions` permite obtener las posiciones más recientes de un dispositivo.

C. *Dependencias Principales*

- **org.traccar.model.GroupedModel**: Interfaz que indica si un modelo pertenece a un grupo, utilizada en la subclase `Permission` para determinar la inclusión de grupos.

XXI. Request

A. *Descripción*

La clase `Request` representa una solicitud para ejecutar una consulta en el sistema Traccar. Permite definir las columnas a seleccionar, las condiciones para filtrar los resultados y el orden en que deben ser devueltos los datos.

XXII. Código

```java
/*
 * 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;
    }

}
```

Listing 11. Request

A. *¿Cómo funciona?*

La clase `Request` encapsula tres componentes principales necesarios para ejecutar una consulta:

- **Columnas:** Especifica las columnas que se deben incluir en el resultado de la consulta.
- **Condición:** Define los criterios para filtrar los resultados de la consulta.
- **Orden:** Determina el orden en que se deben devolver los resultados.

B. *Funcionalidades Principales*

- **Constructor con columnas:** `Request(Columns columns)` permite crear una solicitud especificando únicamente las columnas a seleccionar.
- **Constructor con condición:** `Request(Condition condition)` permite crear una solicitud especificando únicamente la condición para filtrar los resultados.
- **Constructor con columnas y condición:** `Request(Columns columns, Condition condition)` permite crear una solicitud especificando las columnas y la condición de filtrado.
- **Constructor con columnas y orden:** `Request(Columns columns, Order order)` permite crear una solicitud especificando las columnas a seleccionar y el orden de los resultados.
- **Constructor completo:** `Request(Columns columns, Condition condition, Order order)` permite crear una solicitud con todas las opciones configuradas.

C. *Métodos Principales*

- **getColumns:** `public Columns getColumns()` retorna las columnas especificadas en la solicitud.
- **getCondition:** `public Condition getCondition()` retorna la condición de filtrado especificada en la solicitud.
- **getOrder:** `public Order getOrder()` retorna el orden de los resultados especificado en la solicitud.

Nuestra API usa la dependencia de: II. BASERESOURCE, y esta misma dependencia usa otras 3 dependencias ya mencionadas, a continuación vamos a explicar estas mismas:

XXIII. PERMISSIONSSERVICE

A. *Descripción*

La clase `PermissionsService` en la API de Traccar se encarga de gestionar la verificación de permisos y restricciones de los usuarios sobre diversas operaciones y entidades dentro del sistema Traccar. Este servicio es crucial para asegurar que los usuarios tengan los permisos necesarios para realizar acciones como modificar dispositivos, ejecutar comandos o gestionar otros usuarios.

XXIV. CÓDIGO

```
/*
 * Copyright 2022 - 2023 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.security;

import com.google.inject.servlet.RequestScoped;
import org.traccar.model.BaseModel;
import org.traccar.model.Calendar;
import org.traccar.model.Command;
import org.traccar.model.Device;
import org.traccar.model.Group;
import org.traccar.model.GroupedModel;
import org.traccar.model.ManagedUser;
import org.traccar.model.Notification;
import org.traccar.model.Schedulable;
import org.traccar.model.Server;
import org.traccar.model.User;
```

```java
import org.traccar.model.UserRestrictions;
import org.traccar.storage.Storage;
import org.traccar.storage.StorageException;
import org.traccar.storage.query.Columns;
import org.traccar.storage.query.Condition;
import org.traccar.storage.query.Request;

import jakarta.inject.Inject;
import java.util.Objects;

@RequestScoped
public class PermissionsService {

    private final Storage storage;

    private Server server;
    private User user;

    @Inject
    public PermissionsService(Storage storage) {
        this.storage = storage;
    }

    public Server getServer() throws StorageException {
        if (server == null) {
            server = storage.getObject(Server.class, new Request(new Columns.All()));
        }
        return server;
    }

    public User getUser(long userId) throws StorageException {
        if (user == null && userId > 0) {
            if (userId == ServiceAccountUser.ID) {
                user = new ServiceAccountUser();
            } else {
                user = storage.getObject(User.class, new Request(new Columns.All(), new Condition.Equals("id", userId)));
            }
        }
        return user;
    }

    public boolean notAdmin(long userId) throws StorageException {
        return !getUser(userId).getAdministrator();
    }

    public void checkAdmin(long userId) throws StorageException, SecurityException {
        if (!getUser(userId).getAdministrator()) {
            throw new SecurityException("Administrator access required");
        }
    }

    public void checkManager(long userId) throws StorageException, SecurityException {
        if (!getUser(userId).getAdministrator() && getUser(userId).getUserLimit() == 0) {
            throw new SecurityException("Manager access required");
        }
    }

    public interface CheckRestrictionCallback {
        boolean denied(UserRestrictions userRestrictions);
    }

    public void checkRestriction(long userId, CheckRestrictionCallback callback) throws StorageException, SecurityException {
        if (!getUser(userId).getAdministrator() && (callback.denied(getServer()) || callback.denied(getUser(userId)))) {
            throw new SecurityException("Operation restricted");
        }
    }

    public void checkEdit(long userId, Class<?> clazz, boolean addition, boolean skipReadonly) throws StorageException, SecurityException {
        if (!getUser(userId).getAdministrator()) {
            boolean denied = false;
            if (!skipReadonly && (getServer().getReadonly() || getUser(userId).getReadonly())) {
                denied = true;
            } else if (clazz.equals(Device.class)) {
                denied = getServer().getDeviceReadonly() || getUser(userId).getDeviceReadonly()
                        || addition && getUser(userId).getDeviceLimit() == 0;
                if (!denied && addition && getUser(userId).getDeviceLimit() > 0) {
                    int deviceCount = storage.getObjects(Device.class, new Request(
                            new Columns.Include("id"),
                            new Condition.Permission(User.class, userId, Device.class))).size();
                    denied = deviceCount >= getUser(userId).getDeviceLimit();
                }
            } else if (clazz.equals(Command.class)) {
                denied = getServer().getLimitCommands() || getUser(userId).getLimitCommands();
            }
            if (denied) {
                throw new SecurityException("Write access denied");
            }
        }
    }

    public void checkEdit(long userId, BaseModel object, boolean addition, boolean skipReadonly) throws StorageException, SecurityException {
        if (!getUser(userId).getAdministrator()) {
            checkEdit(userId, object.getClass(), addition, skipReadonly);
            if (object instanceof GroupedModel after) {
                if (after.getGroupId() > 0) {
                    GroupedModel before = null;
```

```
                if (!addition) {
                    before = storage.getObject(
                            after.getClass(), new
                            Request(
                                new Columns.Include("
                                    groupId"), new
                                    Condition.Equals("id
                                    ", after.getId())));
                }
                if (before == null || before.
                        getGroupId() != after.
                        getGroupId()) {
                    checkPermission(Group.class,
                            userId, after.getGroupId()
                            );
                }
            }
        }
        if (object instanceof Schedulable after
                ) {
            if (after.getCalendarId() > 0) {
                Schedulable before = null;
                if (!addition) {
                    before = storage.getObject(
                            after.getClass(), new
                            Request(
                                new Columns.Include("
                                    calendarId"), new
                                    Condition.Equals("id
                                    ", object.getId()))
                                    ;
                }
                if (before == null || before.
                        getCalendarId() != after.
                        getCalendarId()) {
                    checkPermission(Calendar.class
                            , userId, after.
                            getCalendarId());
                }
            }
        }
        if (object instanceof Notification
                after) {
            if (after.getCommandId() > 0) {
                Notification before = null;
                if (!addition) {
                    before = storage.getObject(
                            after.getClass(), new
                            Request(
                                new Columns.Include("
                                    commandId"), new
                                    Condition.Equals("id
                                    ", object.getId()))
                                    ;
                }
                if (before == null || before.
                        getCommandId() != after.
                        getCommandId()) {
                    checkPermission(Command.class,
                            userId, after.
                            getCommandId());
                }
            }
        }
    }
}

public void checkUser(long userId, long
        managedUserId) throws StorageException,
        SecurityException {
    if (userId != managedUserId && !getUser(
            userId).getAdministrator()) {
        if (!getUser(userId).getManager()
                || storage.getPermissions(User.
                    class, userId, ManagedUser.
                    class, managedUserId).isEmpty
                    ()) {
            throw new SecurityException("User␣
                access␣denied");
        }
    }
}

public void checkUserUpdate(long userId, User
        before, User after) throws
        StorageException, SecurityException {
    if (before.getAdministrator() != after.
            getAdministrator()
            || before.getDeviceLimit() != after.
                getDeviceLimit()
            || before.getUserLimit() != after.
                getUserLimit()) {
        checkAdmin(userId);
    }
    User user = userId > 0 ? getUser(userId) :
            null;
    if (user != null && user.getExpirationTime
            () != null
            && !Objects.equals(before.
                getExpirationTime(), after.
                getExpirationTime())
            && (after.getExpirationTime() ==
                null
            || user.getExpirationTime().
                compareTo(after.
                getExpirationTime()) < 0)) {
        checkAdmin(userId);
    }
    if (before.getReadonly() != after.
            getReadonly()
            || before.getDeviceReadonly() !=
                after.getDeviceReadonly()
            || before.getDisabled() != after.
                getDisabled()
            || before.getLimitCommands() !=
                after.getLimitCommands()
            || before.getDisableReports() !=
                after.getDisableReports()
            || before.getFixedEmail() != after.
                getFixedEmail()) {
        if (userId == after.getId()) {
            checkAdmin(userId);
        } else if (after.getId() > 0) {
            checkUser(userId, after.getId());
        } else {
            checkManager(userId);
        }
    }
    if (before.getFixedEmail() && !before.
            getEmail().equals(after.getEmail())) {
        checkAdmin(userId);
    }
}

public <T extends BaseModel> void
        checkPermission(
            Class<T> clazz, long userId, long
                objectId) throws StorageException,
                SecurityException {
    if (!getUser(userId).getAdministrator() &&
            !(clazz.equals(User.class) && userId
            == objectId)) {
        var object = storage.getObject(clazz,
                new Request(
                    new Columns.Include("id"),
                    new Condition.And(
```

```
                        new Condition.Equals("id",
                            objectId),
                        new Condition.Permission(
                              User.class, userId,
                                   clazz.equals(User
                                   .class) ?
                                   ManagedUser.class
                                    : clazz))));
            if (object == null) {
                throw new SecurityException(clazz.
                    getSimpleName() + " access
                    denied");
            }
        }
    }

}
```

Listing 12. PermissionsService

A. *¿Cómo funciona?*

`PermissionsService` es un componente inyectado que interactúa con el almacenamiento de datos para validar los permisos del usuario en relación con las acciones que desea realizar. Los métodos principales permiten:

- **Obtener Servidor (getServer):** Recupera la información del servidor si aún no se ha obtenido, utilizando la clase `Storage`.
- **Obtener Usuario (getUser):** Recupera la información de un usuario específico basándose en su ID, permitiendo la validación de permisos personalizados.
- **Verificación de permisos de administrador (checkAdmin):** Comprueba si el usuario tiene permisos de administrador y lanza una excepción de seguridad si no es así.
- **Verificación de permisos de manager (checkManager):** Similar a la verificación de administrador, pero también comprueba límites específicos de gestión de usuarios.
- **Verificación de restricciones (checkRestriction):** Valida si las restricciones del servidor o del usuario impiden la operación solicitada.
- **Verificación de permisos de edición (checkEdit):** Controla si un usuario tiene permitido modificar o añadir entidades como `Device`, `Command`, `Group`, entre otros.

B. *Funcionalidades Principales*

- **Control de acceso:** Antes de realizar cualquier operación, la clase verifica los permisos del usuario sobre la entidad o acción requerida.
- **Manejo de excepciones:** Lanza excepciones de seguridad si se detectan intentos de acceso o modificación sin los permisos adecuados.
- **Integración con el almacenamiento:** Utiliza la clase `Storage` para interactuar con la base de datos y recuperar la información necesaria.

C. *Dependencias Principales*

- **org.traccar.model.BaseModel**: Clase base para los modelos en Traccar, proporcionando una estructura común.
- **org.traccar.model.Device**: Representa un dispositivo de rastreo.
- **org.traccar.model.Group**: Representa un grupo de dispositivos.
- **org.traccar.model.ManagedUser**: Representa un usuario gestionado dentro del sistema.
- **org.traccar.model.Notification**: Representa una notificación en el sistema.
- **org.traccar.model.Schedulable**: Representa una entidad que puede ser programada (como un calendario).
- **org.traccar.model.User**: Representa un usuario del sistema.
- **org.traccar.storage.Storage**: Proporciona acceso al almacenamiento de datos en el sistema.
- **org.traccar.storage.StorageException**: Excepción lanzada en caso de problemas con la base de datos o almacenamiento.
- **org.traccar.storage.query.Columns**: Utilizado para especificar las columnas a recuperar en una consulta.
- **org.traccar.storage.query.Condition**: Utilizado para definir condiciones en las consultas a la base de datos.
- **org.traccar.storage.query.Request**: Clase que encapsula una solicitud de consulta a la base de datos.

XXV. UserPrincipal

A. *Descripción*

La clase `UserPrincipal` en la API de Traccar implementa la interfaz `Principal` y representa

una entidad de usuario autenticada dentro del sistema. Esta clase encapsula la información básica del usuario, como su ID y la fecha de expiración de su sesión o token, lo que es esencial para la gestión de autenticación y autorización en Traccar.

B. *¿Cómo funciona?*

`UserPrincipal` es una implementación de `Principal` que se utiliza principalmente para identificar a los usuarios en el contexto de la seguridad y autenticación. Sus métodos principales son:

- **Constructor (UserPrincipal):** Inicializa un objeto `UserPrincipal` con el ID del usuario y la fecha de expiración.
- **Obtener ID de usuario (getUserId):** Devuelve el ID del usuario asociado con esta instancia de `UserPrincipal`.
- **Obtener Expiración (getExpiration):** Devuelve la fecha de expiración de la sesión o token del usuario.
- **Obtener Nombre (getName):** Método implementado de la interfaz `Principal`, pero en este caso no devuelve un valor significativo, ya que siempre retorna `null`.

C. *Funcionalidades Principales*

- **Identificación de usuario:** Proporciona un mecanismo para identificar de manera única a un usuario en el sistema mediante su ID.
- **Gestión de sesiones:** Almacena la fecha de expiración de la sesión o token, permitiendo que el sistema determine si una sesión es válida.

D. *Dependencias Principales*

- **java.security.Principal**: Interfaz estándar de Java para representar una entidad identificable, como un usuario.
- **java.util.Date**: Clase utilizada para manejar la fecha de expiración de la sesión o token del usuario.

XXVI. Storage

A. *Descripción*

La clase `Storage` en la API de Traccar es una clase abstracta que define la interfaz para interactuar con el almacenamiento de datos del sistema. Esta clase maneja operaciones CRUD (Crear, Leer, Actualizar y Eliminar) sobre los modelos de datos de Traccar, y gestiona las relaciones de permisos entre diferentes entidades.

XXVII. Código

```java
/*
 * 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;

import org.traccar.model.BaseModel;
import org.traccar.model.Permission;
import org.traccar.storage.query.Request;

import java.util.List;

public abstract class Storage {

    public abstract <T> List<T> getObjects(Class<T> clazz, Request request) throws StorageException;

    public abstract <T> long addObject(T entity, Request request) throws StorageException;

    public abstract <T> void updateObject(T entity, Request request) throws StorageException;

    public abstract void removeObject(Class<?> clazz, Request request) throws StorageException;

    public abstract List<Permission> getPermissions(
            Class<? extends BaseModel> ownerClass, long ownerId,
            Class<? extends BaseModel> propertyClass, long propertyId) throws StorageException;

```

```
38      public abstract void addPermission(Permission
            permission) throws StorageException;
39
40      public abstract void removePermission(
            Permission permission) throws
            StorageException;
41
42      public List<Permission> getPermissions(
43              Class<? extends BaseModel> ownerClass,
44              Class<? extends BaseModel>
                    propertyClass) throws
                    StorageException {
45          return getPermissions(ownerClass, 0,
                propertyClass, 0);
46      }
47
48      public <T> T getObject(Class<T> clazz,
            Request request) throws StorageException
            {
49          var objects = getObjects(clazz, request);
50          return objects.isEmpty() ? null : objects.
                get(0);
51      }
52
53  }
```

Listing 13. Storage

A. *¿Cómo funciona?*

`Storage` actúa como un intermediario entre la lógica de negocio y el sistema de persistencia de datos, proporcionando métodos abstractos que deben ser implementados por subclases concretas. Los métodos principales incluyen:

- **Obtener Objetos (getObjects):** Recupera una lista de objetos de una clase específica que cumplen con los criterios especificados en un `Request`.
- **Añadir Objeto (addObject):** Inserta un nuevo objeto en la base de datos y devuelve el ID del objeto creado.
- **Actualizar Objeto (updateObject):** Actualiza un objeto existente en la base de datos basado en los parámetros especificados en un `Request`.
- **Eliminar Objeto (removeObject):** Elimina un objeto de la base de datos basado en la clase y los criterios especificados en un `Request`.
- **Obtener Permisos (getPermissions):** Recupera una lista de permisos entre dos clases de modelo, permitiendo gestionar las relaciones entre entidades.
- **Añadir Permiso (addPermission):** Añade un permiso específico a la base de datos, estableciendo una relación entre dos entidades.
- **Eliminar Permiso (removePermission):** Elimina un permiso existente de la base de datos.
- **Obtener un solo objeto (getObject):** Recupera un único objeto que cumple con los criterios especificados en un `Request`.

B. *Funcionalidades Principales*

- **Gestión de datos:** Proporciona métodos para realizar operaciones CRUD sobre los modelos de Traccar.
- **Gestión de permisos:** Facilita la gestión de permisos y relaciones entre diferentes entidades del sistema.

C. *Dependencias Principales*

- **org.traccar.model.BaseModel**: Clase base para todos los modelos de datos en Traccar.
- **org.traccar.model.Permission**: Representa una relación de permiso entre dos entidades.
- **org.traccar.storage.query.Request**: Clase utilizada para encapsular los criterios de consulta para las operaciones de almacenamiento.
- **org.traccar.storage.StorageException**: Excepción lanzada en caso de problemas con el almacenamiento de datos.

XXVIII. Conclusión y Recomendaciones

A. *Conclusión*

La API de Traccar Position proporciona una interfaz robusta para gestionar los datos de posición de los dispositivos de rastreo. Ofrece funcionalidades esenciales como la recuperación, eliminación y exportación de posiciones en múltiples formatos. La capacidad de filtrar posiciones por ID, dispositivo, y rango de tiempo, junto con la verificación de permisos de usuario, garantiza una gestión efectiva y segura de los datos.

Los métodos de exportación en formatos KML, CSV y GPX permiten una amplia gama de aplicaciones, desde la visualización en Google Earth hasta la manipulación en hojas de cálculo y el uso

en dispositivos GPS. La integración con diferentes proveedores de exportación y el manejo de excepciones relacionadas con el almacenamiento aseguran una experiencia de usuario fluida y confiable.

B. Recomendaciones

- **Optimización de Consultas:** Para mejorar el rendimiento, se recomienda revisar las consultas a la base de datos y asegurar que estén optimizadas, especialmente para grandes volúmenes de datos.
- **Gestión de Permisos:** Asegúrese de que las verificaciones de permisos estén bien implementadas y actualizadas, para evitar accesos no autorizados a los datos.
- **Documentación y Ejemplos:** Proporcione ejemplos prácticos y una documentación detallada sobre el uso de la API, lo que ayudará a los desarrolladores a integrar y utilizar la API de manera más efectiva.
- **Pruebas de Exportación:** Realice pruebas exhaustivas de las funcionalidades de exportación para asegurar que los datos se exporten correctamente en todos los formatos soportados y que los archivos generados sean válidos y utilizables.
- **Manejo de Errores:** Implemente un manejo de errores robusto y claro, proporcionando mensajes de error descriptivos que faciliten la resolución de problemas.