Documentación del Endpoint "Maintenance"

1. Descripción del Endpoint "Maintenance"

Endpoint: Maintenance

Descripción:
El endpoint Maintenance está diseñado para llevar a cabo tareas de mantenimiento dentro del sistema. Estas tareas son esenciales para garantizar el correcto funcionamiento, rendimiento y seguridad de la aplicación. Las funciones principales pueden incluir operaciones como la actualización de bases de datos, la limpieza de datos obsoletos, la realización de copias de seguridad, la restauración de sistemas y la gestión de logs.

El mantenimiento puede ser crítico para la estabilidad del sistema, y este endpoint permite programar o ejecutar de manera manual dichas tareas, asegurando que se realicen de forma controlada y eficiente.

Método HTTP:
El método HTTP utilizado por este endpoint puede variar dependiendo de la operación específica. Las operaciones comunes incluyen:

- POST: Para iniciar una nueva tarea de mantenimiento.
- GET: Para consultar el estado de una tarea en curso o el historial de tareas.
- DELETE: Para cancelar una tarea de mantenimiento programada.

URL:
https://api.example.com/v1/maintenance

2. Composición del Endpoint "Maintenance"

El endpoint "Maintenance" está compuesto por varios componentes críticos que trabajan en conjunto para llevar a cabo las tareas solicitadas. A continuación, se describe la estructura y función de cada uno:

1. Controlador de Mantenimiento (Maintenance Controller):
Este es el núcleo del endpoint, responsable de gestionar las solicitudes entrantes. Actúa como un orquestador que determina qué operación de mantenimiento se debe realizar en función de los parámetros recibidos. Este controlador se encarga de:

- Validar las solicitudes.
- Delegar tareas a los componentes adecuados.
- Manejar excepciones y errores.
- Generar respuestas que indiquen el estado de la operación.

2. Módulo de Tareas de Mantenimiento (Maintenance Tasks Module):
Este módulo alberga las funciones específicas de mantenimiento. Cada función corresponde a una tarea particular, como la limpieza de datos, generación de backups, o actualizaciones de sistema. Algunas de las tareas más comunes son:

- **Cleanup Task:** Elimina datos innecesarios o temporales del sistema, ayudando a mantener la base de datos y el almacenamiento en condiciones óptimas.
- **Backup Task:** Crea copias de seguridad de bases de datos y otros elementos críticos, asegurando la recuperación en caso de fallo.
- **Log Management Task:** Administra los registros del sistema, archivando logs antiguos y limpiando aquellos que ya no son necesarios.

3. Módulo de Programación (Scheduler Module):
Este componente se encarga de manejar la programación de las tareas de mantenimiento. Permite definir cuándo y con qué frecuencia se deben ejecutar las tareas. Soporta cron jobs o intervalos específicos definidos por el usuario. Las principales responsabilidades incluyen:

- Gestionar la cola de tareas programadas.
- Ejecutar tareas en el momento adecuado.
- Reprogramar tareas fallidas o retrasadas.

4. Módulo de Notificaciones (Notification Module):
El módulo de notificaciones es crucial para la supervisión y administración del mantenimiento. Proporciona actualizaciones en tiempo real o periódicas sobre el estado de las tareas de mantenimiento. Las notificaciones pueden ser enviadas a través de

diferentes canales, como correos electrónicos, mensajes en un sistema de mensajería o notificaciones push. Este módulo:

- Informa a los administradores sobre el inicio, éxito o fallo de una tarea.
- Genera alertas si una tarea crítica no se ha completado correctamente.
- Permite la configuración de umbrales para recibir notificaciones solo en ciertos escenarios.

5. Base de Datos y Logs:

El endpoint depende de la base de datos para acceder y manipular datos críticos, y de los logs para registrar las operaciones realizadas. Las interacciones comunes incluyen:

- Lectura de datos necesarios para la ejecución de tareas (datos antiguos para la tarea de limpieza).
- Escritura de logs que documentan el progreso y resultado de las tareas.
- Actualización de registros en caso de cambios realizados durante las operaciones de mantenimiento.

3. Request

Parámetros de entrada:

El endpoint acepta varios parámetros que permiten personalizar y controlar las tareas de mantenimiento. Aquí se describen los parámetros clave:

Nombre	Tipo	Descripción	Obligatorio
task	String	Nombre de la tarea de mantenimiento a ejecutar (e.g., "cleanup", "backup")	Sí
schedule	String	Cron expression para programar la tarea	No
force	Bool	Forzar la ejecución inmediata, ignorando la programación existente	No
notify	Bool	Si se debe enviar una notificación al completar la tarea	No

Ejemplo de Request:


```
POST /v1/maintenance HTTP/1.1
Host: api.example.com
Content-Type: application/json
Authorization: Bearer <token>

{
  "task": "backup",
  "schedule": "0 3 * * *",
  "force": false,
  "notify": true
}
```

4. Response

Códigos de respuesta:

Código	Descripción	Ejemplo de respuesta
200	Tarea de mantenimiento completada con éxito	{"status": "success", "message": "Backup completed successfully"}
400	Parámetros inválidos en la solicitud	{"status": "error", "message": "Invalid task specified"}
401	No autorizado	{"status": "error", "message": "Unauthorized access"}
404	Tarea de mantenimiento no encontrada	{"status": "error", "message": "Task not found"}
500	Error del servidor al ejecutar la tarea	{"status": "error", "message": "Internal server error"}

Ejemplo de Response exitoso:

```
{
  "status": "success",
```

```
  "message": "Backup completed successfully"
}
```

5. Autenticación y Autorización

El endpoint "Maintenance" está diseñado para ser accedido únicamente por usuarios con privilegios administrativos. Esto asegura que solo personal autorizado pueda ejecutar o programar tareas que puedan afectar el sistema en su conjunto.

Tipo de Autenticación:
El acceso está protegido mediante tokens JWT. El token debe incluir los roles necesarios para ejecutar las operaciones de mantenimiento.

Roles o permisos necesarios:

- **Administrador del sistema:** Permisos completos para ejecutar, programar y cancelar cualquier tarea de mantenimiento.
- **Operador de mantenimiento:** Permisos limitados, puede ejecutar tareas pero no modificarlas o cancelarlas.

6. Manejo de Errores

El manejo de errores es crítico para el endpoint "Maintenance", dado que las tareas que realiza pueden tener un impacto significativo en el sistema. Se incluyen varios niveles de manejo de errores:

- **Validación de entrada:** Antes de ejecutar cualquier tarea, se validan los parámetros para asegurar que sean correctos y completos.
- **Registro de errores:** Todos los errores se registran en los logs para permitir un análisis posterior.
- **Retries:** Dependiendo de la tarea, puede configurarse un número de reintentos automáticos en caso de fallos temporales.
- **Notificaciones de error:** Si una tarea falla de manera crítica, se envía una notificación inmediata a los administradores.

7. Ejemplos de Uso

Ejemplo 1: Programar una tarea de limpieza de logs

```
POST /v1/maintenance HTTP/1.1
Content-Type: application/json
Authorization: Bearer <token>

{
  "task": "log_cleanup",
  "schedule": "0 0 * * 0",  // Todos los domingos a medianoche
  "force": false,
  "notify": true
}
```

Ejemplo 2: Ejecutar una copia de seguridad inmediata

```
POST /v1/maintenance HTTP/1.1
Content-Type: application/json
Authorization: Bearer <token>

{
  "task": "backup",
  "force": true,
  "notify": true
}
```

8. Consideraciones de Seguridad

Dado que el endpoint "Maintenance" maneja operaciones críticas, la seguridad es un aspecto clave a considerar:

- **Cifrado:** Todas las comunicaciones con este endpoint deben realizarse a través de HTTPS para proteger los datos en tránsito.
- **Rate Limiting:** Implementar limitaciones de frecuencia para prevenir abusos o ataques de denegación de servicio (DoS).
- **Validación estricta:** Validación robusta de entradas para evitar inyecciones de código, comandos, o datos maliciosos.
- **Auditoría:** Registrar todas las operaciones de mantenimiento en logs de auditoría, incluyendo quién ejecutó la tarea y cuándo.

9. Evaluación para Dockerización

Análisis de la composición del Endpoint:

El endpoint "Maintenance" se compone de varios módulos que pueden operar de manera relativamente independiente, lo que lo convierte en un buen candidato para la dockerización. Aquí se analiza cada componente clave:

- **Controlador de Mantenimiento:** Este componente actúa como el punto de entrada principal y puede beneficiarse de ser ejecutado en un contenedor independiente, especialmente si se requiere escalar el servicio de mantenimiento sin impactar otras partes de la aplicación.
- **Módulo de Tareas de Mantenimiento:** Como cada tarea de mantenimiento puede ser intensiva en recursos (e.g., backups), tener este módulo en un contenedor permite asignar los recursos adecuados y aislar estas operaciones para evitar que afecten el rendimiento general del sistema.
- **Módulo de Programación:** La capacidad de programar tareas de manera aislada asegura que las operaciones críticas ocurran en el tiempo adecuado, y la dockerización facilita la gestión de estos con jobs de forma más controlada y predecible.
- **Módulo de Notificaciones:** Aunque no es necesariamente intensivo en recursos, tener este módulo dockerizado permite la flexibilidad de conectarse a diferentes sistemas de notificaciones o incluso migrar entre soluciones de mensajería sin afectar el resto de la aplicación.

Argumentación:

Candidato para Dockerización:
El endpoint "Maintenance" es altamente modular, lo que lo hace adecuado para ser dockerizado. Las ventajas incluyen:

- **Aislamiento:** Dockerizar este endpoint permitiría ejecutar tareas críticas de mantenimiento en un entorno controlado y aislado, reduciendo el riesgo de que errores o sobrecargas impacten el sistema principal.
- **Escalabilidad Independiente:** Se puede escalar el servicio de mantenimiento de forma independiente del resto de la aplicación, lo que es especialmente útil si ciertas tareas requieren más recursos durante períodos específicos.
- **Entorno Consistente:** Al encapsular todo el entorno necesario para la ejecución de tareas en un contenedor, se asegura que las tareas se ejecuten de manera consistente en diferentes entornos (desarrollo, prueba, producción).
- **Despliegue Simplificado:** Docker permite empaquetar y desplegar el servicio de mantenimiento de forma sencilla, facilitando actualizaciones y asegurando que se respeten las configuraciones necesarias.

Importaciones de Maintenance

En esta documentación, se detallan las principales importaciones y componentes clave relacionados con el endpoint "Maintenance" dentro del sistema. El objetivo es proporcionar una comprensión completa de cómo cada importación contribuye a la funcionalidad general del endpoint, permitiendo su integración y ejecución eficiente. A través de un análisis detallado, se desglosan los distintos módulos y clases utilizados, destacando su rol en la gestión de tareas de mantenimiento críticas para la estabilidad y seguridad del sistema.

Importaciones

1. **org.traccar.api**
    - ExtendedObjectResource
2. **org.traccar.model**
    - Maintenance
    - BaseModel
    - Device
    - Group

    - User
    - GroupedModel
3. **org.traccar.storage**
    - StorageException
    - QueryIgnore
    - StorageName
4. **org.traccar.storage.query**
    - Columns
    - Condition
    - Request
5. **org.traccar.helper**
    - Hashing

Detalle de las Importaciones

1. org.traccar.api.ExtendedObjectResource

Código:

```
/*
 * Copyright 2017 - 2022 Anton Tananaev (anton@traccar.org)
 * Copyright 2017 Andrey Kunitsyn (andrey@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.model.BaseModel;
import org.traccar.model.Device;
import org.traccar.model.Group;
import org.traccar.model.User;
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.ws.rs.GET;
import jakarta.ws.rs.QueryParam;
import java.util.Collection;
import java.util.LinkedList;

public class ExtendedObjectResource<T extends BaseModel> extends
BaseObjectResource<T> {

    public ExtendedObjectResource(Class<T> baseClass) {
        super(baseClass);
    }

    @GET
    public Collection<T> get(
            @QueryParam("all") boolean all, @QueryParam("userId") long userId,
            @QueryParam("groupId") long groupId, @QueryParam("deviceId") long
deviceId) throws StorageException {

        var conditions = new LinkedList<Condition>();
```

```
        if (all) {
            if (permissionsService.notAdmin(getUserId())) {
                conditions.add(new Condition.Permission(User.class, getUserId(),
baseClass));
            }
        } else {
            if (userId == 0) {
                conditions.add(new Condition.Permission(User.class, getUserId(),
baseClass));
            } else {
                permissionsService.checkUser(getUserId(), userId);
                conditions.add(new Condition.Permission(User.class, userId,
baseClass).excludeGroups());
            }
        }

        if (groupId > 0) {
            permissionsService.checkPermission(Group.class, getUserId(), groupId);
            conditions.add(new Condition.Permission(Group.class, groupId,
baseClass).excludeGroups());
        }
        if (deviceId > 0) {
            permissionsService.checkPermission(Device.class, getUserId(), deviceId);
            conditions.add(new Condition.Permission(Device.class, deviceId,
baseClass).excludeGroups());
        }

        return storage.getObjects(baseClass, new Request(new Columns.All(),
Condition.merge(conditions)));
    }

}
```

Descripción:

- ExtendedObjectResource<T> es una clase genérica que extiende BaseObjectResource. Esta clase sirve como base para crear recursos RESTful que operan sobre objetos de tipo BaseModel.

Funcionalidad:

- **CRUD Operaciones:** Permite realizar operaciones CRUD sobre los objetos del tipo especificado (T).
- **Filtrado por Condiciones:** Implementa un método GET que soporta el filtrado de objetos mediante condiciones como el ID del usuario, ID del grupo o ID del dispositivo.
- **Seguridad y Permisos:** Utiliza servicios de permisos (permissionsService) para validar que el usuario tiene los permisos adecuados antes de ejecutar las operaciones solicitadas.

Base:

- Este diseño sigue los principios de la programación genérica en Java, lo que permite reutilizar la misma lógica para múltiples tipos de objetos.
- Integra servicios de seguridad y permisos directamente en las operaciones CRUD, asegurando que solo los usuarios autorizados puedan realizar ciertas acciones.

2. org.traccar.model.Maintenance

Código:

```
/*
 * Copyright 2018 Anton Tananaev (anton@traccar.org)
 * Copyright 2018 Andrey Kunitsyn (andrey@traccar.org)
```

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

import org.traccar.storage.StorageName;

@StorageName("tc_maintenances")
public class Maintenance extends ExtendedModel {

    private String name;

    public String getName() {
        return name;
    }

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

    private String type;
```

```
    public String getType() {
        return type;
    }

    public void setType(String type) {
        this.type = type;
    }

    private double start;

    public double getStart() {
        return start;
    }

    public void setStart(double start) {
        this.start = start;
    }

    private double period;

    public double getPeriod() {
        return period;
    }

    public void setPeriod(double period) {
        this.period = period;
    }

}
```

Descripción:

- El modelo Maintenance representa una tarea de mantenimiento en el sistema, almacenando información sobre el tipo de mantenimiento, el nombre, el inicio y el periodo.

Funcionalidad:

- **Almacenamiento Persistente:** Está anotado con @StorageName("tc_maintenances"), lo que indica que los datos de mantenimiento se guardan en la tabla tc_maintenances en la base de datos.
- **Propiedades Clave:**
  - name: El nombre de la tarea de mantenimiento.
  - type: El tipo de mantenimiento (e.g., cambio de aceite, revisión general).
  - start: Punto de inicio para realizar el mantenimiento (e.g., kilometraje inicial).
  - period: El intervalo de tiempo o distancia en el que se debe repetir el mantenimiento.

Base:

- Utiliza la herencia para extender ExtendedModel, lo que le otorga funcionalidades adicionales de modelo base.
- Los métodos get y set siguen las convenciones de JavaBeans, facilitando la manipulación y acceso a los datos.

3. org.traccar.model.BaseModel

Código:

```
/*
 * Copyright 2017 - 2022 Anton Tananaev (anton@traccar.org)
 * Copyright 2017 Andrey Kunitsyn (andrey@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
o...
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package org.traccar.model;

public class BaseModel {

    private long id;

    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

}
```

Descripción:

- BaseModel es una clase base para otros modelos en el sistema. Define una propiedad id que sirve como identificador único para todas las instancias que heredan de esta clase.

Funcionalidad:

- **Identificador Único:** Proporciona un id que puede ser utilizado para identificar de manera única cada objeto en la base de datos.
- **Herencia:** Es la clase base de la que derivan otros modelos, como Maintenance, para mantener una estructura consistente y facilitar la gestión de identificadores.

Base:

- La estructura simple de esta clase asegura que todos los modelos derivados tengan un campo de identificación único, lo que es esencial para operaciones de CRUD y manejo de persistencia.

4. org.traccar.model.Device

Código:

```
/*
 * 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";

private String status;

@QueryIgnore
public String getStatus() {
   return status != null ? status : STATUS_OFFLINE;
}

public void setStatus(String status) {
   this.status = status != null ? status.trim() : null;
}

private Date lastUpdate;

@QueryIgnore
```

```
public Date getLastUpdate() {
    return this.lastUpdate;
}

public void setLastUpdate(Date lastUpdate) {
    this.lastUpdate = lastUpdate;
}

private long positionId;

@QueryIgnore
public long getPositionId() {
    return positionId;
}

public void setPositionId(long positionId) {
    this.positionId = positionId;
}

private String phone;

public String getPhone() {
    return phone;
}

public void setPhone(String phone) {
    this.phone = phone != null ? phone.trim() : null;
}

private String model;

public String getModel() {
    return model;
```

```
}

public void setModel(String model) {
    this.model = model;
}

private String contact;

public String getContact() {
    return contact;
}

public void setContact(String contact) {
    this.contact = contact;
}

private String category;

public String getCategory() {
    return category;
}

public void setCategory(String category) {
    this.category = category;
}

private boolean disabled;

@Override
public boolean getDisabled() {
    return disabled;
}
```

```
@Override
public void setDisabled(boolean disabled) {
    this.disabled = disabled;
}

private Date expirationTime;

@Override
public Date getExpirationTime() {
    return expirationTime;
}

@Override
public void setExpirationTime(Date expirationTime) {
    this.expirationTime = expirationTime;
}

private boolean motionStreak;

@QueryIgnore
@JsonIgnore
public boolean getMotionStreak() {
    return motionStreak;
}

@JsonIgnore
public void setMotionStreak(boolean motionStreak) {
    this.motionStreak = motionStreak;
}

private boolean motionState;

@QueryIgnore
```

```
@JsonIgnore
public boolean getMotionState() {
    return motionState;
}

@JsonIgnore
public void setMotionState(boolean motionState) {
    this.motionState = motionState;
}

private Date motionTime;

@QueryIgnore
@JsonIgnore
public Date getMotionTime() {
    return motionTime;
}

@JsonIgnore
public void setMotionTime(Date motionTime) {
    this.motionTime = motionTime;
}

private double motionDistance;

@QueryIgnore
@JsonIgnore
public double getMotionDistance() {
    return motionDistance;
}

@JsonIgnore
public void setMotionDistance(double motionDistance) {
```

```
    this.motionDistance = motionDistance;
}

private boolean overspeedState;

@QueryIgnore
@JsonIgnore
public boolean getOverspeedState() {
    return overspeedState;
}

@JsonIgnore
public void setOverspeedState(boolean overspeedState) {
    this.overspeedState = overspeedState;
}

private Date overspeedTime;

@QueryIgnore
@JsonIgnore
public Date getOverspeedTime() {
    return overspeedTime;
}

@JsonIgnore
public void setOverspeedTime(Date overspeedTime) {
    this.overspeedTime = overspeedTime;
}

private long overspeedGeofenceId;

@QueryIgnore
@JsonIgnore
```

```
    public long getOverspeedGeofenceId() {
        return overspeedGeofenceId;
    }

    @JsonIgnore
    public void setOverspeedGeofenceId(long overspeedGeofenceId) {
        this.overspeedGeofenceId = overspeedGeofenceId;
    }

}
```

Descripción:

- Device es un modelo que representa un dispositivo en el sistema de rastreo (e.g., un dispositivo GPS).

Funcionalidad:

- **Propiedades Clave:**
  - name: Nombre del dispositivo.
  - uniqueId: Identificador único del dispositivo, utilizado para diferenciarlo de otros dispositivos.
  - status: Estado actual del dispositivo (online, offline, unknown).
  - lastUpdate: Fecha de la última actualización del dispositivo.
- **Interfaz Disableable:** Define métodos para habilitar o deshabilitar el dispositivo.
- **Interfaz Schedulable:** Permite asociar un dispositivo a un calendario para la programación de eventos.
- **Persistencia:** La anotación @StorageName("tc_devices") indica que este modelo se guarda en la tabla tc_devices.

Base:

- La implementación de interfaces como Disableable y Schedulable permite extender las capacidades del modelo para interactuar con otras partes del sistema.
- Las anotaciones de persistencia aseguran que el modelo se maneje de manera consistente con el esquema de la base de datos.

5. org.traccar.model.Group

Código:

```
/*
 * Copyright 2016 - 2018 Anton Tananaev (anton@traccar.org)
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package org.traccar.model;

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

```
@StorageName("tc_groups")
public class Group extends GroupedModel {

    private String name;

    public String getName() {
        return name;
    }

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

}
```

Descripción:

- Group es un modelo que representa un grupo de dispositivos o usuarios en el sistema.

Funcionalidad:

- **Propiedades Clave:**
  - name: El nombre del grupo.
- **Herencia:** Extiende GroupedModel, lo que permite que un grupo esté asociado con otros objetos que también heredan de GroupedModel.
- **Persistencia:** Se almacena en la tabla tc_groups, como lo indica la anotación @StorageName.

Base:

- La herencia de GroupedModel permite que Group herede comportamientos y propiedades que son comunes a todos los modelos agrupados.

- Facilita la gestión de dispositivos y usuarios agrupados, permitiendo organizar mejor los recursos en el sistema.

6. org.traccar.model.User

Código:

```
/*
 * Copyright 2013 - 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
o...
 * 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.apache.commons.lang3.builder.EqualsBuilder;
import org.traccar.storage.QueryIgnore;
import org.traccar.helper.Hashing;
import org.traccar.storage.StorageName;

import java.util.Date;
```

```
import java.util.HashMap;

@StorageName("tc_users")
public class User extends ExtendedModel implements UserRestrictions, Disableable
{

    private String name;

    public String getName() {
        return name;
    }

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

    private String login;

    public String getLogin() {
        return login;
    }

    public void setLogin(String login) {
        this.login = login;
    }

    private String email;

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
```

```
    this.email = email.trim();
}

private String phone;

public String getPhone() {
    return phone;
}

public void setPhone(String phone) {
    this.phone = phone != null ? phone.trim() : null;
}

private boolean readonly;

@Override
public boolean getReadonly() {
    return readonly;
}

public void setReadonly(boolean readonly) {
    this.readonly = readonly;
}

private boolean administrator;

@QueryIgnore
@JsonIgnore
public boolean getManager() {
    return userLimit != 0;
}

public boolean getAdministrator() {
```

```
    return administrator;
}

public void setAdministrator(boolean administrator) {
    this.administrator = administrator;
}

private String map;

public String getMap() {
    return map;
}

public void setMap(String map) {
    this.map = map;
}

private double latitude;

public double getLatitude() {
    return latitude;
}

public void setLatitude(double latitude) {
    this.latitude = latitude;
}

private double longitude;

public double getLongitude() {
    return longitude;
}
```

```
public void setLongitude(double longitude) {
    this.longitude = longitude;
}

private int zoom;

public int getZoom() {
    return zoom;
}

public void setZoom(int zoom) {
    this.zoom = zoom;
}

private String coordinateFormat;

public String getCoordinateFormat() {
    return coordinateFormat;
}

public void setCoordinateFormat(String coordinateFormat) {
    this.coordinateFormat = coordinateFormat;
}

private boolean disabled;

@Override
public boolean getDisabled() {
    return disabled;
}

@Override
public void setDisabled(boolean disabled) {
```

```
    this.disabled = disabled;
}

private Date expirationTime;

@Override
public Date getExpirationTime() {
    return expirationTime;
}

@Override
public void setExpirationTime(Date expirationTime) {
    this.expirationTime = expirationTime;
}

private int deviceLimit;

public int getDeviceLimit() {
    return deviceLimit;
}

public void setDeviceLimit(int deviceLimit) {
    this.deviceLimit = deviceLimit;
}

private int userLimit;

public int getUserLimit() {
    return userLimit;
}

public void setUserLimit(int userLimit) {
    this.userLimit = userLimit;
```

```
}

private boolean deviceReadonly;

@Override
public boolean getDeviceReadonly() {
   return deviceReadonly;
}

public void setDeviceReadonly(boolean deviceReadonly) {
   this.deviceReadonly = deviceReadonly;
}

private boolean limitCommands;

@Override
public boolean getLimitCommands() {
   return limitCommands;
}

public void setLimitCommands(boolean limitCommands) {
   this.limitCommands = limitCommands;
}

private boolean disableReports;

@Override
public boolean getDisableReports() {
   return disableReports;
}

public void setDisableReports(boolean disableReports) {
   this.disableReports = disableReports;
```

```
}

private boolean fixedEmail;

@Override
public boolean getFixedEmail() {
    return fixedEmail;
}

public void setFixedEmail(boolean fixedEmail) {
    this.fixedEmail = fixedEmail;
}

private String poiLayer;

public String getPoiLayer() {
    return poiLayer;
}

public void setPoiLayer(String poiLayer) {
    this.poiLayer = poiLayer;
}

private String totpKey;

public String getTotpKey() {
    return totpKey;
}

public void setTotpKey(String totpKey) {
    this.totpKey = totpKey;
}
```

```
private boolean temporary;

public boolean getTemporary() {
    return temporary;
}

public void setTemporary(boolean temporary) {
    this.temporary = temporary;
}

@QueryIgnore
public String getPassword() {
    return null;
}

@QueryIgnore
public void setPassword(String password) {
    if (password != null && !password.isEmpty()) {
        Hashing.HashingResult hashingResult = Hashing.createHash(password);
        hashedPassword = hashingResult.getHash();
        salt = hashingResult.getSalt();
    }
}

private String hashedPassword;

@JsonIgnore
@QueryIgnore
public String getHashedPassword() {
    return hashedPassword;
}

@QueryIgnore
```

```
    public void setHashedPassword(String hashedPassword) {
        this.hashedPassword = hashedPassword;
    }

    private String salt;

    @JsonIgnore
    @QueryIgnore
    public String getSalt() {
        return salt;
    }

    @QueryIgnore
    public void setSalt(String salt) {
        this.salt = salt;
    }

    public boolean isPasswordValid(String password) {
        return Hashing.validatePassword(password, hashedPassword, salt);
    }

    public boolean compare(User other, String... exclusions) {
        if (!EqualsBuilder.reflectionEquals(this, other, "attributes", "hashedPassword",
"salt")) {
            return false;
        }
        var thisAttributes = new HashMap<>(getAttributes());
        var otherAttributes = new HashMap<>(other.getAttributes());
        for (String exclusion : exclusions) {
            thisAttributes.remove(exclusion);
            otherAttributes.remove(exclusion);
        }
        return thisAttributes.equals(otherAttributes);
```

```
    }
}
```

Descripción:

- User es un modelo que representa a un usuario en el sistema.

Funcionalidad:

- **Propiedades Clave:**
    - name, login, email, phone: Datos de identificación y contacto del usuario.
    - administrator: Indica si el usuario tiene permisos de administrador.
    - deviceLimit, userLimit: Restricciones que limitan la cantidad de dispositivos y usuarios que un usuario puede gestionar.
- **Autenticación:** Implementa métodos para gestionar contraseñas utilizando hashing (Hashing).
- **Persistencia:** Se almacena en la tabla tc_users según la anotación @StorageName.

Base:

- La implementación de funcionalidades como el hashing de contraseñas y la validación de permisos hace que este modelo sea fundamental para la gestión de la seguridad y autenticación en el sistema.

7. org.traccar.storage.StorageException

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
o...
 * 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);
    }

}
```

Descripción:

- StorageException es una excepción personalizada que se utiliza para manejar errores relacionados con la persistencia de datos en el sistema.

Funcionalidad:

- **Captura de Errores:** Proporciona varias sobrecargas del constructor para capturar y reportar diferentes tipos de errores, incluyendo mensajes personalizados y excepciones encadenadas.
- **Integración:** Utilizado en varias partes del sistema para manejar errores que ocurren durante operaciones de almacenamiento en la base de datos.

Base:

- Facilita el manejo de excepciones específicas de la capa de almacenamiento, proporcionando una forma estructurada de manejar y reportar errores en las operaciones de persistencia.

8. org.traccar.storage.QueryIgnore

Código:

```
/*
 * Copyright 2017 Anton Tananaev (anton@traccar.org)
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express
o...
```

```
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package org.traccar.storage;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface QueryIgnore {
}
```

Descripción:

- QueryIgnore es una anotación que se utiliza para excluir métodos o propiedades de ser considerados durante las consultas a la base de datos.

Funcionalidad:

- **Optimización de Consultas:** Al usar esta anotación, se pueden excluir ciertos métodos o propiedades que no son necesarios para ser cargados o evaluados durante una operación de consulta, mejorando así el rendimiento de las consultas.
- **Personalización:** Permite un mayor control sobre qué partes del modelo se incluyen en las consultas, evitando la sobrecarga de información innecesaria.

Base:

- Proporciona un mecanismo de control sobre las consultas a la base de datos, asegurando que solo los datos relevantes sean procesados durante las operaciones de persistencia.

9. org.traccar.storage.StorageName

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
o...
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package org.traccar.storage;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
public @interface StorageName {
   String value();
}
```

Descripción:

- StorageName es una anotación que se utiliza para especificar el nombre de la tabla de base de datos con la que un modelo está asociado.

Funcionalidad:

- **Mapeo de Tablas:** Permite que un modelo de Java esté asociado con una tabla específica en la base de datos, facilitando la persistencia y recuperación de datos.
- **Consistencia:** Asegura que los modelos se guarden y carguen desde la tabla correcta, lo que es fundamental para la integridad de los datos en el sistema.

Base:

- Esta anotación es clave para el ORM (Object-Relational Mapping) del sistema, estableciendo la conexión entre los modelos de Java y la estructura de la base de datos subyacente.

10. org.traccar.storage.query.Columns

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
o...
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package org.traccar.storage.query;

import org.traccar.storage.QueryIgnore;

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

public abstract class Columns {

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

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

```
    return columns;
  }

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

  public static class Include extends Columns {
    private final List<String> columns;

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

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

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

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

    @Override
    public List<String> getColumns(Class<?> clazz, String type) {
      return getAllColumns(clazz, type).stream()
```

```
                .filter(column -> !columns.contains(column))
                .collect(Collectors.toList());
        }
    }

}
```

Descripción:

- Columns es una clase abstracta que define qué columnas deben incluirse o excluirse en una consulta a la base de datos.

Funcionalidad:

- **Subclases All, Include, Exclude:** Proporcionan diferentes estrategias para seleccionar columnas en una consulta:
    - All: Incluye todas las columnas disponibles.
    - Include: Incluye solo las columnas especificadas.
    - Exclude: Excluye las columnas especificadas.
- **Reflexión:** Utiliza reflexión para obtener y manipular los métodos del modelo Java y determinar qué columnas deben incluirse en la consulta.

Base:

- Esta clase es fundamental para personalizar las consultas a la base de datos, permitiendo un alto grado de flexibilidad en cómo se seleccionan y manipulan los datos.

11. org.traccar.storage.query.Condition

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
o...
 * 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;
        }
    }

}
```

Descripción:

- Condition es una interfaz que define condiciones que se aplican a consultas en la base de datos.

Funcionalidad:

- **Subclases Equals, Compare, Between, Or, And, Binary, Permission, LatestPositions:** Estas subclases proporcionan diferentes maneras de definir y combinar condiciones para consultas:
    - Equals: Filtra registros donde una columna tiene un valor específico.
    - Between: Filtra registros donde una columna está entre dos valores.

    - Or, And, Binary: Combinan múltiples condiciones usando operadores lógicos.
    - Permission: Filtra registros basados en permisos asociados con un usuario o grupo.
- **Combinación de Condiciones:** El método estático merge combina una lista de condiciones en una sola condición usando operadores lógicos.

Base:

- Proporciona una estructura para construir consultas complejas en la base de datos, permitiendo filtrar, combinar, y especificar condiciones basadas en permisos y otros criterios lógicos.

12. org.traccar.storage.query.Request

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
o...
 * 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;
    }

}
```

Descripción:

- Request encapsula una consulta a la base de datos, combinando columnas, condiciones y órdenes en un solo objeto.

Funcionalidad:

- **Personalización de Consultas:** Permite construir consultas detalladas especificando qué columnas incluir, bajo qué condiciones, y en qué orden se deben devolver los resultados.
- **Integración con Columns y Condition:** Utiliza las clases Columns y Condition para definir los detalles de la consulta, asegurando que solo los datos relevantes se seleccionen y se devuelvan.

Base:

- Esta clase proporciona una forma estructurada de construir consultas SQL complejas, permitiendo un control preciso sobre cómo se recuperan los datos de la base de datos.

13. org.traccar.helper.Hashing

Código:

```
/*
 * Copyright 2015 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
o...
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package org.traccar.helper;

import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.PBEKeySpec;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;
import java.security.spec.InvalidKeySpecException;

public final class Hashing {

    public static final int ITERATIONS = 1000;
    public static final int SALT_SIZE = 24;
    public static final int HASH_SIZE = 24;

    private static SecretKeyFactory factory;
    static {
```

```
        try {
            factory = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1");
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
    }

    public static class HashingResult {

        private final String hash;
        private final String salt;

        public HashingResult(String hash, String salt) {
            this.hash = hash;
            this.salt = salt;
        }

        public String getHash() {
            return hash;
        }

        public String getSalt() {
            return salt;
        }
    }

    private Hashing() {
    }

    private static byte[] function(char[] password, byte[] salt) {
        try {
            PBEKeySpec spec = new PBEKeySpec(password, salt, ITERATIONS,
HASH_SIZE * Byte.SIZE);
```

```
            return factory.generateSecret(spec).getEncoded();
        } catch (InvalidKeySpecException e) {
            throw new SecurityException(e);
        }
    }

    private static final SecureRandom RANDOM = new SecureRandom();

    public static HashingResult createHash(String password) {
        byte[] salt = new byte[SALT_SIZE];
        RANDOM.nextBytes(salt);
        byte[] hash = function(password.toCharArray(), salt);
        return new HashingResult(
                DataConverter.printHex(hash),
                DataConverter.printHex(salt));
    }

    public static boolean validatePassword(String password, String hashHex, String
saltHex) {
        byte[] hash = DataConverter.parseHex(hashHex);
        byte[] salt = DataConverter.parseHex(saltHex);
        return slowEquals(hash, function(password.toCharArray(), salt));
    }

    /**
     * Compares two byte arrays in length-constant time. This comparison method
     * is used so that password hashes cannot be extracted from an on-line
     * system using a timing attack and then attacked off-line.
     */
    private static boolean slowEquals(byte[] a, byte[] b) {
        int diff = a.length ^ b.length;
        for (int i = 0; i < a.length && i < b.length; i++) {
            diff |= a[i] ^ b[i];
```

```
        }
        return diff == 0;
    }

}
```

Descripción:

- Hashing es una clase de utilidad que proporciona métodos para crear y validar contraseñas utilizando algoritmos de hashing.

Funcionalidad:

- **Creación de Hashes:** Utiliza el algoritmo PBKDF2WithHmacSHA1 para crear hashes seguros de contraseñas.
- **Generación de Sal (Salt):** Crea un valor salt aleatorio que se combina con la contraseña antes de aplicar el hashing, mejorando la seguridad contra ataques de diccionario.
- **Validación de Contraseñas:** Permite validar si una contraseña dada coincide con un hash almacenado, utilizando comparación constante en tiempo (slowEquals) para evitar ataques de temporización.

Base:

- La clase sigue las mejores prácticas de seguridad para el manejo de contraseñas, proporcionando un método robusto para asegurar que las contraseñas de los usuarios estén protegidas adecuadamente en el sistema.

Conclusión

El endpoint "Maintenance" está diseñado para manejar tareas críticas que aseguran el correcto funcionamiento y estabilidad del sistema. A través de una estructura modular y bien organizada, este endpoint permite realizar operaciones como la limpieza de datos, backups, y gestión de logs de manera eficiente y segura.

Los componentes clave que componen el endpoint, desde el controlador de mantenimiento hasta los módulos de notificaciones y programación, están diseñados para operar de manera independiente pero coordinada, lo que facilita la escalabilidad y la gestión de las tareas de mantenimiento. Este modularidad también lo convierte en un excelente candidato para la dockerización, lo que proporcionaría beneficios adicionales en términos de aislamiento, escalabilidad, y despliegue consistente en diferentes entornos.

El endpoint Maintenance es una pieza fundamental dentro del sistema, encargado de gestionar las tareas de mantenimiento que garantizan el buen funcionamiento y la integridad de los datos del sistema. Este endpoint, junto con las clases e importaciones relacionadas, sigue una arquitectura bien definida que aprovecha el poder de la herencia y la modularidad en Java para facilitar la implementación, mantenimiento y expansión del sistema.

Resumen de la Funcionalidad del Endpoint "Maintenance"

- **Propósito:**
  El endpoint "Maintenance" permite gestionar tareas de mantenimiento en el sistema, como la creación, actualización, y eliminación de registros de mantenimiento. Esto incluye la capacidad de programar y monitorizar estas tareas para asegurar que se ejecuten de manera oportuna.
- **Composición:**
  Este endpoint extiende la clase ExtendedObjectResource, lo que le proporciona capacidades avanzadas para realizar operaciones CRUD sobre el modelo Maintenance. Además, utiliza servicios de permisos para validar el acceso de los usuarios y asegurar que las operaciones se realicen de manera segura y controlada.
- **Importaciones Clave:**
  - **org.traccar.api.ExtendedObjectResource:** Proporciona una base para realizar operaciones CRUD y gestionar permisos de acceso.
  - **org.traccar.model.Maintenance:** Modelo que representa una tarea de mantenimiento, incluyendo atributos como nombre, tipo, inicio y periodo.

- **Otras Importaciones:** Como BaseModel, Device, Group, User, entre otras, que facilitan la estructura y funcionalidad del sistema en su conjunto.

Evaluación y Recomendaciones

El diseño modular y extensible del endpoint "Maintenance" lo hace un excelente candidato para ser dockerizado, permitiendo un despliegue y escalabilidad más efectivos. La separación de responsabilidades y la clara definición de los modelos y controladores aseguran que el sistema sea mantenible y fácil de expandir en el futuro.

Referencias

La información provista en esta documentación se ha basado en el análisis del código proporcionado y otros apartados, código el cual pertenece a un sistema de rastreo y gestión de dispositivos construido sobre la plataforma Traccar. Este análisis incluye la revisión de las clases e importaciones en el código fuente y se apoya en principios y prácticas estándar de desarrollo en Java.

- Código fuente proporcionado: https://github.com/Ooops-seb/DockerTraccar.git
- Richardson, L., & Ruby, S. (2007). *RESTful Web Services*. O'Reilly Media.
- Grassi, J., Garcia, M., & Fenton, J. (2020). *Digital Identity Guidelines: Authentication and Lifecycle Management (NIST SP 800-63B)*. National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-63b
- Merkel, D. (2014). Docker: Lightweight Linux Containers for Consistent Development and Deployment. *Linux Journal, 2014*(239), Article 2.