Laboratorio 3::DockerTraccar::Calendars

1st Joel Cuenca
Ingeniería de Software
Universidad de las Fuerzas Armadas ESPE
Machala - Ecuador
ajcuenca4@espe.edu.ec

I. Resumen

Abstract

Este documento proporciona una explicación detallada de varias clases en Java que forman parte de un sistema de gestión de datos. Se aborda la funcionalidad de la clase abstracta Columns, que proporciona métodos para obtener listas de columnas basadas en la reflexión. Se detallan las subclases All, Include y Exclude, que implementan diferentes estrategias para manejar columnas. Además, se examina la clase StorageException, que actúa como una excepción personalizada para errores relacionados con el almacenamiento, y la clase User, que representa un modelo de usuario con diversas propiedades y métodos, incluyendo la gestión de contraseñas mediante la clase Hashing. Finalmente, se describen las anotaciones QueryIgnore y StorageName, que proporcionan información adicional para la gestión de consultas y almacenamiento.

II. Introducción

Docker Traccar es una versión contenedorizada de Traccar, un software de seguimiento GPS de código abierto. Docker simplifica la instalación y administración de Traccar al encapsular todas las dependencias del sistema y la aplicación en un contenedor. Esto facilita la portabilidad, escalabilidad y gestión de la aplicación en diferentes entornos, ya que Docker garantiza que Traccar funcione de manera consistente sin importar el sistema operativo subyacente. [1]

En el contexto de Traccar o sistemas similares, un endpoint calendars generalmente se refiere a un punto de acceso en una API que maneja las interacciones relacionadas con calendarios. Este endpoint podría permitir la creación, modificación, eliminación y consulta de calendarios asociados con diferentes eventos o entidades en la plataforma. [2]

Por ejemplo, en un sistema de gestión de dispositivos de rastreo como Traccar, los calendarios pueden ser utilizados para programar eventos, como tareas de mantenimiento de los dispositivos, programar alertas, o definir horarios de operación para vehículos. Este endpoint proporciona la funcionalidad necesaria para interactuar con estos calendarios mediante solicitudes HTTP.

El uso de calendars en una API facilita la automatización y gestión eficiente de eventos temporales, lo que es crucial para aplicaciones que dependen de la programación y el seguimiento en tiempo real. [3]

III. Materiales y Métodos

Computadora 1

- **Nombre del Dispositivo:** Laptop MSI (DESKTOP-F8DOUN9)
- **Procesador:** 11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz 2.30 GHz
- **RAM instalada:** 16.0 GB
- **Maquina Virtual:** VirtualBox
- **ISO:** Debian
- **Conexión a Internet**

IV. Metodología

Descripción del Endpoint

El endpoint `calendars` permite gestionar los recursos de calendario en la API. A través de este endpoint, los usuarios pueden realizar operaciones CRUD (Crear, Leer, Actualizar y Eliminar) sobre los calendarios. A continuación, se detallan las operaciones disponibles y su uso.

Operaciones Disponibles

A. 1. Obtener todos los calendarios

- **Método HTTP**: `GET`
- **URL**: `https://api.ejemplo.com/calendars`
- **Descripción**: Recupera una lista de todos los calendarios disponibles en el sistema.
- **Parámetros de Consulta**:
  - `startDate`: Fecha de inicio para filtrar los calendarios por su fecha de inicio.
  - `endDate`: Fecha de finalización para filtrar los calendarios por su fecha de finalización.
  - `name`: Nombre del calendario para realizar una búsqueda por nombre.
- **Respuesta Exitosa**:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
    {
        "id": 1,
        "name": "Calendario de Eventos",
        "startDate": "2024-01-01",
        "endDate": "2024-12-31"
    },
    {
        "id": 2,
        "name": "Calendario de Tareas",
        "startDate": "2024-02-01",
        "endDate": "2024-11-30"
    }
]
```

B. 2. Crear un nuevo calendario

- **Método HTTP**: `POST`
- **URL**: `https://api.ejemplo.com/calendars`
- **Descripción**: Crea un nuevo calendario. Los detalles del calendario deben ser proporcionados en el cuerpo de la solicitud en formato JSON.
- **Cuerpo de la Solicitud**:

```
{
    "name": "Nuevo Calendario",
    "startDate": "2024-03-01",
    "endDate": "2024-10-31"
}
```

- **Respuesta Exitosa**:

```
HTTP/1.1 201 Created
Content-Type: application/json

{
    "id": 3,
    "name": "Nuevo Calendario",
    "startDate": "2024-03-01",
    "endDate": "2024-10-31"
}
```

C. 3. Obtener un calendario específico

- **Método HTTP**: `GET`
- **URL**: `https://api.ejemplo.com/calendars/id`
- **Descripción**: Recupera los detalles del calendario especificado por el `id` en la ruta.
- **Respuesta Exitosa**:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "name": "Calendario de Eventos",
    "startDate": "2024-01-01",
    "endDate": "2024-12-31"
}
```

- **Respuesta de Error**:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "error": "Calendario no encontrado"
}
```

D. 4. Actualizar un calendario específico

- **Método HTTP**: `PUT`
- **URL**: `https://api.ejemplo.com/calendars/id`
- **Descripción**: Actualiza el calendario especificado por el `id` en la ruta. Los datos actualizados deben ser proporcionados en el cuerpo de la solicitud en formato JSON.
- **Cuerpo de la Solicitud**:

```
{
    "name": "Calendario Actualizado",
    "startDate": "2024-03-01",
    "endDate": "2024-11-30"
}
```

- **Respuesta Exitosa**:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "id": 1,
    "name": "Calendario Actualizado",
    "startDate": "2024-03-01",
    "endDate": "2024-11-30"
}
```

E. 5. Eliminar un calendario específico

- **Método HTTP**: `DELETE`
- **URL**: `https://api.ejemplo.com/calendars/id`
- **Descripción**: Elimina el calendario especificado por el `id` en la ruta.
- **Respuesta Exitosa**:

```
HTTP/1.1 204 No Content
```

- **Respuesta de Error**:

```
HTTP/1.1 404 Not Found
Content-Type: application/json

{
    "error": "Calendario no encontrado"
}
```

Ejemplos de Solicitudes

F. Ejemplo de Solicitud GET para obtener todos los calendarios

```
GET https://api.ejemplo.com/calendars?startDate=2024-01-01&endDate=2024-12-31
```

G. Ejemplo de Solicitud POST para crear un nuevo calendario

```
POST https://api.ejemplo.com/calendars
Content-Type: application/json

{
    "name": "Nuevo Calendario",
    "startDate": "2024-03-01",
    "endDate": "2024-10-31"
}
```

H. Ejemplo de Solicitud PUT para actualizar un calendario

```
PUT https://api.ejemplo.com/calendars/1
Content-Type: application/json

{
    "name": "Calendario Actualizado",
    "startDate": "2024-03-01",
    "endDate": "2024-11-30"
}
```

I. Ejemplo de Solicitud DELETE para eliminar un calendario

```
DELETE https://api.ejemplo.com/calendars/1
```

Código Java: CalendarResource

```java
/*
 * package org.traccar.api.resource;
 */
package org.traccar.api.resource;

import jakarta.ws.rs.Consumes;
import jakarta.ws.rs.Path;
import jakarta.ws.rs.Produces;
import jakarta.ws.rs.core.MediaType;

import org.traccar.api.SimpleObjectResource;
import org.traccar.model.Calendar;

@Path("calendars")
@Produces(MediaType.APPLICATION_JSON)
@Consumes(MediaType.APPLICATION_JSON)
public class CalendarResource extends SimpleObjectResource<Calendar> {

    public CalendarResource() {
        super(Calendar.class);
    }

}
```

Explicación Detallada del Código

Este código es una clase de Java llamada `CalendarResource`, que forma parte del paquete `org.traccar.api.resource`. Está implementada dentro del proyecto Traccar, una plataforma de rastreo GPS. La clase utiliza anotaciones JAX-RS para exponer un endpoint REST que gestiona calendarios (`Calendar`). A continuación, se detalla cada componente del código para entender su propósito y funcionamiento:

1. Paquete (`package org.traccar.api.resource`)

Descripción: Este código está en el paquete `org.traccar.api.resource`. Los paquetes en Java agrupan clases relacionadas, lo que ayuda a organizar el código y a evitar conflictos de nombres. El paquete `org.traccar.api.resource` contiene los recursos API del sistema Traccar, que exponen las operaciones REST para interactuar con los datos. Aquí, `CalendarResource` es uno de esos recursos.

2. Imports

`jakarta.ws.rs.Consumes` y `jakarta.ws.rs.Produces`:

Descripción: Estas anotaciones son parte del estándar JAX-RS para definir los formatos de datos que la clase puede consumir y producir en sus operaciones REST. `Consumes` indica que el recurso puede recibir datos en un formato específico, y `Produces` define el formato en el que devolverá las respuestas.

`jakarta.ws.rs.Path`:

Descripción: Esta anotación especifica la URL relativa que debe coincidir con el endpoint REST. En este caso, `@Path("calendars")` indica que la clase `CalendarResource` estará disponible en la ruta `/calendars`.

`jakarta.ws.rs.core.MediaType`:

Descripción: `MediaType` es una clase que define los tipos de contenido MIME, como `MediaType.APPLICATION_JSON`, que representa el formato JSON. Este tipo de contenido es común en APIs RESTful, ya que JSON es legible tanto para máquinas como para humanos.

`org.traccar.api.SimpleObjectResource`:

Descripción: `SimpleObjectResource` es una clase base genérica que proporciona funcionalidades comunes para gestionar objetos a través de un API REST. En este caso, `CalendarResource` hereda de `SimpleObjectResource<Calendar>`, lo que le permite manejar operaciones CRUD sobre objetos de tipo `Calendar`.

`org.traccar.model.Calendar`:

Descripción: `Calendar` es el modelo de datos que representa un calendario en el sistema Traccar. Los calendarios se utilizan para definir eventos y periodos de tiempo que pueden estar asociados a dispositivos o alertas en el sistema.

3. Clase `CalendarResource`

Esta clase es un recurso RESTful que expone operaciones sobre los calendarios en el sistema Traccar. Está diseñada para manejar las solicitudes HTTP relacionadas con el recurso `Calendar`.

Herencia: `extends SimpleObjectResource<Calendar>`:

Descripción: `CalendarResource` extiende de `SimpleObjectResource<Calendar>`. Esto significa que hereda toda la funcionalidad básica de `SimpleObjectResource` para manejar operaciones CRUD (Create, Read, Update, Delete) sobre el objeto `Calendar`. Al heredar de esta clase, `CalendarResource` puede automáticamente gestionar las operaciones REST estándar sin necesidad de definir cada una explícitamente.

Ejemplo: Si un cliente envía una solicitud HTTP `GET` a `/calendars`, el sistema invocará el método de recuperación de calendarios proporcionado por `SimpleObjectResource`. Del mismo modo, una solicitud `POST` creará un nuevo calendario, y `PUT` o `DELETE` actualizará o eliminará un calendario existente.

Anotación `@Path("calendars")`:

Descripción: Define la ruta base de este recurso RESTful. Las solicitudes dirigidas a `http://servidor/api/calendars` serán manejadas por esta clase. En combinación con los métodos HTTP (`GET, POST, PUT, DELETE`), esto permite que los clientes interactúen con los calendarios del sistema.

Anotación `@Produces(MediaType.APPLICATION_JSON)`:

Descripción: Indica que las respuestas de este recurso serán en formato JSON. Es decir, cuando un cliente realiza una solicitud a este endpoint, la respuesta será enviada en formato JSON. JSON es un formato de intercambio de datos muy utilizado en aplicaciones web y móviles por su simplicidad y legibilidad.

Anotación `@Consumes(MediaType.APPLICATION_JSON)`:

Descripción: Indica que este recurso consumirá solicitudes en formato JSON. Esto significa que cuando un cliente envíe datos a este recurso (por ejemplo, un calendario nuevo), los datos deben estar en formato JSON para ser procesados correctamente.

4. Constructor `CalendarResource()`

Descripción: El constructor de la clase `CalendarResource` simplemente invoca al constructor de la clase base (`SimpleObjectResource`) y le pasa el tipo `Calendar.class`. Este paso es crucial porque `SimpleObjectResource` es una clase genérica, y el constructor necesita saber qué tipo de objeto va a gestionar. En este caso, la clase está configurada para manejar objetos de tipo `Calendar`.

5. Propósito General de `CalendarResource`

Descripción: El propósito principal de la clase `CalendarResource` es proporcionar un punto de acceso RESTful para gestionar calendarios dentro del sistema Traccar. Los calendarios pueden contener eventos que definen periodos de tiempo importantes para los dispositivos o alertas en el sistema. Al utilizar esta clase, los clientes pueden crear, leer, actualizar y eliminar calendarios.

Interacción con el Sistema:

El sistema Traccar utiliza calendarios para gestionar horarios de funcionamiento, eventos y otros periodos de tiempo que pueden estar vinculados a dispositivos GPS. Por ejemplo, un calendario podría definir cuándo un dispositivo debe estar activo o cuándo una alerta debe ser enviada.

La clase `CalendarResource` permite a los administradores del sistema crear y gestionar estos calendarios a través de una API REST, lo que facilita la integración con aplicaciones móviles, paneles web u otros sistemas.

6. Ejemplo de Uso

Operación de Creación (POST):

Un cliente puede enviar una solicitud HTTP `POST` a `/calendars` con un cuerpo en formato JSON que contenga los detalles de un nuevo calendario. La clase `CalendarResource`, a través de la herencia de `SimpleObjectResource`, manejará esta solicitud y creará un nuevo objeto `Calendar` en la base de datos.

Operación de Recuperación (GET):

Si un cliente envía una solicitud HTTP `GET` a `/calendars`, `CalendarResource` devolverá una lista de calendarios en formato JSON. El formato JSON facilita que otros sistemas o aplicaciones consuman estos datos.

Operación de Actualización (PUT):

El cliente puede enviar una solicitud HTTP `PUT` a `/calendars/{id}` para actualizar un calendario existente. La solicitud debe incluir el ID del calendario a actualizar y un cuerpo JSON con los cambios que se desean realizar.

Operación de Eliminación (DELETE):

Una solicitud HTTP `DELETE` a `/calendars/{id}` eliminará el calendario correspondiente en el sistema.

CÓDIGO JAVA: CALENDAR

```java
package org.traccar.model;

import com.fasterxml.jackson.annotation.JsonIgnore;
import net.fortuna.ical4j.data.CalendarBuilder;
import net.fortuna.ical4j.data.ParserException;
import net.fortuna.ical4j.model.Period;
import net.fortuna.ical4j.model.component.CalendarComponent;
import net.fortuna.ical4j.model.component.VEvent;
import org.traccar.storage.QueryIgnore;
import org.traccar.storage.StorageName;

import java.io.ByteArrayInputStream;
import java.io.IOException;
import java.time.Duration;
import java.time.Instant;
import java.util.Date;
import java.util.Set;
import java.util.stream.Collectors;

@StorageName("tc_calendars")
public class Calendar extends ExtendedModel {

    private String name;

    public String getName() {
        return name;
    }

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

```
    }

    private byte[] data;

    public byte[] getData() {
        return data;
    }

    public void setData(byte[] data) throws IOException, ParserException {
        CalendarBuilder builder = new CalendarBuilder();
        calendar = builder.build(new ByteArrayInputStream(data));
        this.data = data;
    }

    private net.fortuna.ical4j.model.Calendar calendar;

    @QueryIgnore
    @JsonIgnore
    public net.fortuna.ical4j.model.Calendar getCalendar() {
        return calendar;
    }

    public Set<Period<Instant>> findPeriods(Date date) {
        if (calendar != null) {
            var period = new Period<>(date.toInstant(), Duration.ZERO);
            return calendar.<VEvent>getComponents(CalendarComponent.VEVENT).stream()
                    .flatMap(c -> c.<Instant>calculateRecurrenceSet(period).stream())
                    .collect(Collectors.toUnmodifiableSet());
        } else {
            return Set.of();
        }
    }

    public boolean checkMoment(Date date) {
        return !findPeriods(date).isEmpty();
    }

}
```

Explicación Detallada del Código

El código pertenece a la clase `Calendar` dentro del paquete `org.traccar.model`. Esta clase es parte del modelo de datos del sistema Traccar y está diseñada para representar un calendario que puede contener eventos y manejar recursiones. A continuación, se explica cada sección del código.

1. Paquete e Imports

`package org.traccar.model;`

Descripción: La clase `Calendar` está en el paquete `org.traccar.model`, que agrupa las clases de modelo de datos del sistema Traccar.

Imports:

- `com.fasterxml.jackson.annotation.JsonIgnore`: Esta anotación se usa para evitar que el campo anotado sea serializado o deserializado en JSON.
- `net.fortuna.ical4j.data.CalendarBuilder`: Permite construir un calendario a partir de datos de entrada en formato iCalendar.
- `net.fortuna.ical4j.data.ParserException`: Excepción lanzada cuando ocurre un error al analizar datos de iCalendar.
- `net.fortuna.ical4j.model.Period`: Representa un período de tiempo en un calendario.
- `net.fortuna.ical4j.model.component.CalendarComponent`: Componente base de iCalendar (por ejemplo, eventos).
- `net.fortuna.ical4j.model.component.VEvent`: Representa un evento dentro de un calendario iCalendar.
- `org.traccar.storage.QueryIgnore`: Anotación que indica que el campo debe ignorarse durante las consultas a la base de datos.
- `org.traccar.storage.StorageName`: Define el nombre de la tabla de almacenamiento en la base de datos.
- `java.util.Date`, `java.time.Instant`, `java.util.Set`, `java.util.stream.Collectors`: Manejo de fechas, periodos y colecciones en Java.

2. Anotaciones de Clase

`@StorageName("tc_calendars"):`

Descripción: Define el nombre de la tabla en la base de datos como `tc_calendars`. Esta anotación es parte del sistema de persistencia del proyecto y asegura que los datos de esta clase se almacenen en la tabla correspondiente.

3. Atributos de la Clase

`private String name;:`

Descripción: Este atributo almacena el nombre del calendario. Tiene los métodos `getName()` y `setName()` para acceder y modificar el valor.

`private byte[] data;:`

Descripción: Este atributo almacena los datos del calendario en formato de bytes. El método `setData(byte[] data)` convierte estos bytes en un objeto de tipo `Calendar` utilizando la clase `CalendarBuilder`.

`private net.fortuna.ical4j.model.Calendar calendar;:`

Descripción: Este es el objeto `Calendar` que representa el calendario deserializado. Se inicializa cuando se establece el atributo `data`.

4. Métodos de la Clase

`public String getName()` y `public void setName(String name):`

Descripción: Estos métodos son los accesores y mutadores (getters y setters) para el atributo `name`. Permiten leer y modificar el nombre del calendario.

`public byte[] getData()` y `public void setData(byte[] data) throws IOException, ParserException`:

Descripción: El método `getData()` devuelve los datos en formato de bytes del calendario. El método `setData(byte[] data)` convierte los bytes en un objeto `Calendar` y los almacena en el atributo `calendar`. Si ocurre un error durante la conversión, se lanzan las excepciones `IOException` o `ParserException`.

`public net.fortuna.ical4j.model.Calendar getCalendar():`

Descripción: Este método devuelve el objeto `Calendar` deserializado. Está anotado con `@QueryIgnore` y `@JsonIgnore`, lo que significa que no se incluirá en las consultas de la base de datos ni en las serializaciones JSON.

`public Set<Period<Instant» findPeriods(Date date):`

Descripción: Este método busca los periodos en el calendario que coinciden con una fecha específica. Utiliza el objeto `calendar` para buscar eventos (`VEvent`) y calcula los periodos de recurrencia de estos eventos en función de la fecha proporcionada.

`public boolean checkMoment(Date date):`

Descripción: Este método verifica si hay eventos en el calendario que coincidan con una fecha específica. Retorna `true` si hay eventos que coinciden y `false` en caso contrario.

5. Propósito General de la Clase `Calendar`

La clase `Calendar` en el sistema Traccar representa un calendario que puede contener eventos. Los calendarios se almacenan en la base de datos en la tabla `tc_calendars`. Esta clase proporciona métodos para manipular y consultar los eventos en un calendario, manejando tanto la serialización/deserialización de datos como la búsqueda de periodos y eventos.

CÓDIGO JAVA

A continuación, se muestra el código Java para la clase `SimpleObjectResource`:

```
1  package org.traccar.api;
2
3  import org.traccar.model.BaseModel;
4  import org.traccar.model.User;
5  import org.traccar.storage.StorageException;
6  import org.traccar.storage.query.Columns;
7  import org.traccar.storage.query.Condition;
8  import org.traccar.storage.query.Request;
9
10 import jakarta.ws.rs.GET;
11 import jakarta.ws.rs.QueryParam;
12 import java.util.Collection;
13 import java.util.LinkedList;
14
15 public class SimpleObjectResource<T extends BaseModel> extends BaseObjectResource<T> {
16
17     public SimpleObjectResource(Class<T> baseClass) {
```

```
18          super(baseClass);
19      }
20
21      @GET
22      public Collection<T> get(
23              @QueryParam("all") boolean all, @QueryParam("userId") long userId) throws StorageException {
24
25          var conditions = new LinkedList<Condition>();
26
27          if (all) {
28              if (permissionsService.notAdmin(getUserId())) {
29                  conditions.add(new Condition.Permission(User.class, getUserId(), baseClass));
30              }
31          } else {
32              if (userId == 0) {
33                  userId = getUserId();
34              } else {
35                  permissionsService.checkUser(getUserId(), userId);
36              }
37              conditions.add(new Condition.Permission(User.class, userId, baseClass));
38          }
39
40          return storage.getObjects(baseClass, new Request(new Columns.All(), Condition.merge(conditions)));
41      }
42
43  }
```

Explicación Detallada del Código

El código anterior define una clase Java genérica llamada `SimpleObjectResource`, que extiende de `BaseObjectResource`. Esta clase es parte del sistema Traccar, una plataforma de rastreo GPS, y proporciona una API RESTful para gestionar objetos que heredan de `BaseModel`.

J. Paquete e Imports

- `package org.traccar.api`: Este código se encuentra en el paquete `org.traccar.api`, que contiene las clases relacionadas con la API del sistema Traccar.
- `import org.traccar.model.BaseModel`: `BaseModel` es una clase que sirve como base para todos los modelos en el sistema Traccar.
- `import org.traccar.model.User`: Importa el modelo de usuario, ya que es necesario para verificar permisos.
- `import org.traccar.storage.*`: Estos imports permiten el uso de clases relacionadas con el almacenamiento, como excepciones y consultas.
- `import jakarta.ws.rs.*`: Estas clases son parte de JAX-RS y se utilizan para definir recursos y métodos HTTP en la API REST.

K. Clase `SimpleObjectResource`

`SimpleObjectResource` es una clase genérica que maneja objetos del tipo `T`, donde `T` extiende de `BaseModel`. Esto permite que la clase gestione cualquier tipo de modelo del sistema Traccar que herede de `BaseModel`.

1) Constructor: El constructor de la clase recibe un parámetro de tipo `Class<T>` que representa la clase del modelo que será gestionada por esta instancia. Este constructor simplemente llama al constructor de la clase base, `BaseObjectResource`, con el mismo parámetro.

2) Método `get()`: Este método es el que responde a las solicitudes HTTP GET y devuelve una colección de objetos del tipo `T`. El método recibe dos parámetros:

- `all`: Un booleano que indica si se deben obtener todos los objetos o solo aquellos pertenecientes a un usuario específico.
- `userId`: El ID del usuario cuyos objetos se desean recuperar.

Dependiendo del valor de `all` y `userId`, el método construye una lista de condiciones que se utilizarán para filtrar los resultados de la consulta.

L. Condiciones de Permiso

El método agrega condiciones basadas en los permisos del usuario:

- Si `all` es `true`, pero el usuario no es administrador, se agrega una condición que limita los resultados a aquellos que el usuario tiene permiso para ver.
- Si `all` es `false`, se verifica que el `userId` proporcionado coincida con el del usuario actual, y se agrega una condición para filtrar los resultados por usuario.

Finalmente, el método realiza una consulta al almacenamiento utilizando las condiciones construidas y devuelve los objetos que cumplen con ellas.

Código Java

A continuación, se muestra el código Java para la clase `Request`:

```java
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;
    }

}
```

Explicación Detallada del Código

El código anterior define una clase Java llamada `Request` que se utiliza para encapsular los parámetros de una consulta a la base de datos en el sistema Traccar. A continuación, se detalla cada componente del código:

M. Paquete e Imports

- `package org.traccar.storage.query`: Este código se encuentra en el paquete `org.traccar.storage.query`, que contiene clases relacionadas con las consultas y el almacenamiento de datos en el sistema Traccar.

N. Clase `Request`

La clase `Request` tiene tres atributos principales que definen una consulta:

- `columns`: Un objeto de tipo `Columns` que especifica qué columnas se deben incluir en el resultado de la consulta.
- `condition`: Un objeto de tipo `Condition` que define las condiciones de filtrado para la consulta.
- `order`: Un objeto de tipo `Order` que determina el orden de los resultados de la consulta.

O. Constructores

La clase `Request` tiene varios constructores sobrecargados que permiten crear instancias de `Request` con diferentes combinaciones de parámetros:

- `Request(Columns columns)`: Crea una instancia de `Request` con columnas especificadas, sin condición ni orden.
- `Request(Condition condition)`: Crea una instancia de `Request` con condición especificada, sin columnas ni orden.
- `Request(Columns columns, Condition condition)`: Crea una instancia de `Request` con columnas y condición especificadas, sin orden.
- `Request(Columns columns, Order order)`: Crea una instancia de `Request` con columnas y orden especificados, sin condición.
- `Request(Columns columns, Condition condition, Order order)`: Crea una instancia de `Request` con columnas, condición y orden especificados.

P. Métodos Getters

La clase proporciona métodos para acceder a los atributos privados:

- `getColumns()`: Devuelve el objeto `Columns` asociado con esta solicitud.
- `getCondition()`: Devuelve el objeto `Condition` asociado con esta solicitud.
- `getOrder()`: Devuelve el objeto `Order` asociado con esta solicitud.

Q. Propósito General

La clase `Request` sirve como una estructura para encapsular todos los parámetros necesarios para realizar una consulta a la base de datos. Permite especificar qué columnas incluir en los resultados, qué condiciones aplicar para filtrar los datos y cómo ordenar los resultados. Esta clase facilita la construcción y el manejo de consultas de una manera organizada y flexible.

CÓDIGO JAVA

A continuación, se muestra el código Java para la clase `GroupedModel`:

```java
package org.traccar.model;

public class GroupedModel extends ExtendedModel {

    private long groupId;

    public long getGroupId() {
        return groupId;
    }

    public void setGroupId(long groupId) {
        this.groupId = groupId;
    }

}
```

EXPLICACIÓN DETALLADA DEL CÓDIGO

El código anterior define una clase Java llamada `GroupedModel` que extiende la clase `ExtendedModel`. Esta clase se utiliza para representar un modelo que tiene un identificador de grupo asociado. A continuación, se detalla cada componente del código:

R. Paquete e Imports

- `package org.traccar.model`: Este código está en el paquete `org.traccar.model`, que contiene las clases de modelos en el sistema Traccar.

S. Clase `GroupedModel`

La clase `GroupedModel` hereda de `ExtendedModel`, lo que significa que hereda todas las características y comportamientos de `ExtendedModel`.

1) Atributo `groupId`:

- `private long groupId`: Es un atributo de tipo `long` que representa el identificador del grupo. El modificador `private` indica que este atributo solo puede ser accedido y modificado dentro de la clase `GroupedModel`.

2) Métodos Getters y Setters:

- `public long getGroupId()`: Es un método público que devuelve el valor actual de `groupId`.
- `public void setGroupId(long groupId)`: Es un método público que establece el valor del atributo `groupId`.

T. Propósito General

La clase `GroupedModel` se utiliza para representar objetos que tienen un identificador de grupo. Hereda de `ExtendedModel`, lo que sugiere que puede estar asociada con otras funcionalidades y atributos definidos en `ExtendedModel`. Los métodos getter y setter proporcionan acceso controlado al atributo `groupId`, permitiendo leer y modificar el identificador del grupo según sea necesario.

CÓDIGO JAVA

A continuación se presenta el código Java para la interfaz `Condition`:

```java
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;
```

```
132         private final Class<?> propertyClass;
133         private final long propertyId;
134         private final boolean excludeGroups;
135 
136         private Permission(
137                 Class<?> ownerClass, long ownerId, Class<?> propertyClass, long propertyId, boolean
                        excludeGroups) {
138             this.ownerClass = ownerClass;
139             this.ownerId = ownerId;
140             this.propertyClass = propertyClass;
141             this.propertyId = propertyId;
142             this.excludeGroups = excludeGroups;
143         }
144 
145         public Permission(Class<?> ownerClass, long ownerId, Class<?> propertyClass) {
146             this(ownerClass, ownerId, propertyClass, 0, false);
147         }
148 
149         public Permission(Class<?> ownerClass, Class<?> propertyClass, long propertyId) {
150             this(ownerClass, 0, propertyClass, propertyId, false);
151         }
152 
153         public Permission excludeGroups() {
154             return new Permission(this.ownerClass, this.ownerId, this.propertyClass, this.propertyId,
                        true);
155         }
156 
157         public Class<?> getOwnerClass() {
158             return ownerClass;
159         }
160 
161         public long getOwnerId() {
162             return ownerId;
163         }
164 
165         public Class<?> getPropertyClass() {
166             return propertyClass;
167         }
168 
169         public long getPropertyId() {
170             return propertyId;
171         }
172 
173         public boolean getIncludeGroups() {
174             boolean ownerGroupModel = GroupedModel.class.isAssignableFrom(ownerClass);
175             boolean propertyGroupModel = GroupedModel.class.isAssignableFrom(propertyClass);
176             return (ownerGroupModel || propertyGroupModel) && !excludeGroups;
177         }
178     }
179 
180     class LatestPositions implements Condition {
181         private final long deviceId;
182 
183         public LatestPositions(long deviceId) {
184             this.deviceId = deviceId;
185         }
186 
187         public LatestPositions() {
188             this(0);
189         }
190 
191         public long getDeviceId() {
192             return deviceId;
193         }
194     }
195 
196 }
```

Explicación Detallada del Código

El código define una interfaz `Condition` y sus diversas implementaciones para representar diferentes tipos de condiciones en consultas. A continuación, se detalla cada componente:

U. Método Estático `merge`

El método estático `merge(List<Condition> conditions)` combina una lista de condiciones en una sola condición compuesta usando la lógica `AND` entre ellas.

V. Clases Internas

1) Clase `Equals`: La clase `Equals` extiende `Compare` y representa una condición de igualdad en una consulta.

2) Clase `Compare`: La clase `Compare` define una condición de comparación que incluye una columna, un operador, una variable y un valor. Esta clase es utilizada para condiciones que comparan valores en columnas específicas.

3) Clase `Between`: La clase `Between` representa una condición donde el valor de una columna debe estar entre dos valores.

4) Clases `Or` y `And`: Las clases `Or` y `And` representan condiciones lógicas binarias que combinan dos condiciones usando los operadores `OR` y `AND`, respectivamente.

5) Clase `Binary`: La clase `Binary` es una clase base para `Or` y `And`, que maneja condiciones binarias generales.

6) Clase `Permission`: La clase `Permission` gestiona las condiciones relacionadas con los permisos de acceso, incluyendo la validación de grupos y propietarios de objetos.

7) Clase `LatestPositions`: La clase `LatestPositions` representa una condición que filtra por el último dispositivo registrado.

CÓDIGO JAVA

A continuación se presenta el código Java para la clase `Columns`:

```java
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());
        }
    }

}
```

EXPLICACIÓN DETALLADA DEL CÓDIGO

La clase `Columns` es una clase abstracta utilizada para obtener una lista de columnas de un modelo basado en métodos de reflexión. Esta clase define varios métodos y clases internas para obtener columnas de diferentes maneras.

W. Método Abstracto `getColumns`

El método abstracto `getColumns(Class<?> clazz, String type)` debe ser implementado por las subclases para devolver una lista de nombres de columnas basadas en la clase y el tipo especificado.

X. Método Protegido `getAllColumns`

El método protegido `getAllColumns(Class<?> clazz, String type)` obtiene todos los nombres de las columnas a partir de los métodos de la clase dada. Filtra los métodos basados en el prefijo del nombre (por ejemplo, "get" o "set"), el número de parámetros y la presencia de la anotación `QueryIgnore`.

Y. Clases Internas

1) Clase `All`: La clase `All` extiende `Columns` y devuelve todas las columnas obtenidas por `getAllColumns`.

2) Clase `Include`: La clase `Include` permite especificar explícitamente una lista de columnas a incluir. Implementa `getColumns` para devolver la lista de columnas proporcionada en el constructor.

3) Clase `Exclude`: La clase `Exclude` permite especificar una lista de columnas a excluir. Implementa `getColumns` para devolver todas las columnas menos aquellas especificadas en el constructor.

CÓDIGO JAVA

A continuación se presenta el código Java para la clase `StorageException`:

```
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);
    }

}
```

EXPLICACIÓN DETALLADA DEL CÓDIGO

La clase `StorageException` es una excepción personalizada que extiende la clase `Exception` en Java. Se utiliza para manejar errores específicos relacionados con el almacenamiento en el sistema.

Z. Constructores

La clase `StorageException` proporciona tres constructores:

- `StorageException(String message)`: Este constructor permite crear una excepción con un mensaje descriptivo que describe el error ocurrido.
- `StorageException(Throwable cause)`: Este constructor permite crear una excepción con una causa específica, que es otra excepción que causó esta excepción.
- `StorageException(String message, Throwable cause)`: Este constructor permite crear una excepción con un mensaje descriptivo y una causa específica.

Cada uno de estos constructores llama al constructor correspondiente de la clase `Exception` con los parámetros apropiados para asegurar que la excepción pueda ser creada y utilizada de manera adecuada en el manejo de errores.

CÓDIGO JAVA

A continuación se presenta el código Java para la clase `User`:

```java
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;
    }
```

```
131
132     private boolean disabled;
133
134     @Override
135     public boolean getDisabled() {
136         return disabled;
137     }
138
139     @Override
140     public void setDisabled(boolean disabled) {
141         this.disabled = disabled;
142     }
143
144     private Date expirationTime;
145
146     @Override
147     public Date getExpirationTime() {
148         return expirationTime;
149     }
150
151     @Override
152     public void setExpirationTime(Date expirationTime) {
153         this.expirationTime = expirationTime;
154     }
155
156     private int deviceLimit;
157
158     public int getDeviceLimit() {
159         return deviceLimit;
160     }
161
162     public void setDeviceLimit(int deviceLimit) {
163         this.deviceLimit = deviceLimit;
164     }
165
166     private int userLimit;
167
168     public int getUserLimit() {
169         return userLimit;
170     }
171
172     public void setUserLimit(int userLimit) {
173         this.userLimit = userLimit;
174     }
175
176     private boolean deviceReadonly;
177
178     @Override
179     public boolean getDeviceReadonly() {
180         return deviceReadonly;
181     }
182
183     public void setDeviceReadonly(boolean deviceReadonly) {
184         this.deviceReadonly = deviceReadonly;
185     }
186
187     private boolean limitCommands;
188
189     @Override
190     public boolean getLimitCommands() {
191         return limitCommands;
192     }
193
194     public void setLimitCommands(boolean limitCommands) {
195         this.limitCommands = limitCommands;
196     }
197
198     private boolean disableReports;
199
200     @Override
201     public boolean getDisableReports() {
202         return disableReports;
203     }
204
```

```
205     public void setDisableReports(boolean disableReports) {
206         this.disableReports = disableReports;
207     }
208 
209     private boolean fixedEmail;
210 
211     @Override
212     public boolean getFixedEmail() {
213         return fixedEmail;
214     }
215 
216     public void setFixedEmail(boolean fixedEmail) {
217         this.fixedEmail = fixedEmail;
218     }
219 
220     private String poiLayer;
221 
222     public String getPoiLayer() {
223         return poiLayer;
224     }
225 
226     public void setPoiLayer(String poiLayer) {
227         this.poiLayer = poiLayer;
228     }
229 
230     private String totpKey;
231 
232     public String getTotpKey() {
233         return totpKey;
234     }
235 
236     public void setTotpKey(String totpKey) {
237         this.totpKey = totpKey;
238     }
239 
240     private boolean temporary;
241 
242     public boolean getTemporary() {
243         return temporary;
244     }
245 
246     public void setTemporary(boolean temporary) {
247         this.temporary = temporary;
248     }
249 
250     @QueryIgnore
251     public String getPassword() {
252         return null;
253     }
254 
255     @QueryIgnore
256     public void setPassword(String password) {
257         if (password != null && !password.isEmpty()) {
258             Hashing.HashingResult hashingResult = Hashing.createHash(password);
259             hashedPassword = hashingResult.getHash();
260             salt = hashingResult.getSalt();
261         }
262     }
263 
264     private String hashedPassword;
265 
266     @JsonIgnore
267     @QueryIgnore
268     public String getHashedPassword() {
269         return hashedPassword;
270     }
271 
272     @QueryIgnore
273     public void setHashedPassword(String hashedPassword) {
274         this.hashedPassword = hashedPassword;
275     }
276 
277     private String salt;
278 
```

```
    @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);
    }

}
```

Explicación Detallada del Código

La clase `User` extiende `ExtendedModel` e implementa las interfaces `UserRestrictions` y `Disableable`. Esta clase representa un modelo de usuario con diversas propiedades y métodos asociados.

. *Atributos y Métodos*

- `name`, `login`, `email`, `phone`: Propiedades básicas del usuario, con métodos getter y setter correspondientes. La propiedad `email` se limpia de espacios en blanco y `phone` se establece en `null` si es necesario.
- `readonly`, `administrator`, `disabled`, `deviceReadonly`, `limitCommands`, `disableReports`, `fixedEmail`, `temporary`: Propiedades booleanas que representan diferentes configuraciones y restricciones del usuario.
- `map`, `latitude`, `longitude`, `zoom`, `coordinateFormat`, `poiLayer`, `totpKey`: Propiedades adicionales para el usuario que incluyen configuraciones específicas del mapa y autenticación de dos factores.
- `expirationTime`, `deviceLimit`, `userLimit`: Propiedades relacionadas con la duración y límites del usuario.
- `hashedPassword`, `salt`: Propiedades para el manejo seguro de contraseñas, con métodos para establecer y validar contraseñas.

. *Anotaciones y Métodos Especiales*

- `@StorageName("tc_users")`: $Anotacinqueespecificaelnombredealmacenamientoparalaentidaddeusuario.$`@QueryIgnore` $Anotacionesparaignorarciertosmtodosypropiedadesenconsultasyserializacin JSON.$
- `isPasswordValid(String password)`: Método que valida una contraseña comparando con la contraseña hasheada y la sal.
- `compare(User other, String... exclusions)`: Método para comparar el usuario actual con otro usuario, ignorando ciertas propiedades.

Código Java

A continuación se presenta el código Java para la clase `Hashing`:

```
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;

```

```java
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;
```

```
    }

}
```

Explicación Detallada del Código

La clase `Hashing` proporciona utilidades para el hash de contraseñas utilizando el algoritmo PBKDF2 con HMAC-SHA1.

. *Constantes*

- `ITERATIONS`: Número de iteraciones para la función de derivación de clave.
- `SALT_SIZE`: Tamaño del vector de sal en bytes.
- `HASH_SIZE`: Tamaño del hash en bytes.

. *Inicialización*

El bloque estático inicializa la `SecretKeyFactory` utilizando el algoritmo `PBKDF2WithHmacSHA1`. Si el algoritmo no está disponible, se imprime una traza de la excepción.

. *Clase Interna `HashingResult`*

- `hash`: Hash de la contraseña.
- `salt`: Sal utilizada en el proceso de hashing.
- Métodos: `getHash()` y `getSalt()` para obtener el hash y la sal.

. *Métodos Principales*

- `createHash(String password)`: Crea un hash para la contraseña proporcionada, genera una nueva sal y retorna el resultado en formato hexadecimal.
- `validatePassword(String password, String hashHex, String saltHex)`: Valida la contraseña comparando el hash calculado con el hash proporcionado.
- `slowEquals(byte[] a, byte[] b)`: Compara dos arreglos de bytes en tiempo constante para prevenir ataques de temporización.

. *Método Privado `function`*

Este método utiliza `PBEKeySpec` para generar un hash basado en la contraseña y la sal. La excepción `InvalidKeySpecException` es capturada y envuelta en una `SecurityException`.

Código Java

A continuación se presenta el código Java para la clase `BaseModel`:

```java
package org.traccar.model;

public class BaseModel {

    private long id;

    public long getId() {
        return id;
    }

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

}
```

Explicación Detallada del Código

La clase `BaseModel` es una clase base que contiene un identificador único.

. *Atributo*

- `id`: Un campo de tipo `long` que almacena el identificador único del modelo.

. *Métodos*

- `getId()`: Método getter que devuelve el valor del identificador `id`.
- `setId(long id)`: Método setter que asigna un valor al identificador `id`.

Esta clase es generalmente utilizada como una clase base en una jerarquía de clases, proporcionando un identificador común para todas las clases derivadas.

Código Java

A continuación se presenta el código Java para la anotación `QueryIgnore`:

```java
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 {
}
```

Explicación Detallada del Código

La anotación `QueryIgnore` se utiliza para marcar métodos que deben ser ignorados durante la ejecución de ciertas operaciones de consulta.

. *Componentes de la Anotación*

- `@Target(ElementType.METHOD)`: Especifica que esta anotación se puede aplicar a métodos.
- `@Retention(RetentionPolicy.RUNTIME)`: Indica que la anotación está disponible en tiempo de ejecución. Esto permite que la anotación sea accesible a través de reflexión en tiempo de ejecución.

. *Uso*

La anotación `QueryIgnore` se utiliza para marcar métodos que deben ser excluidos de ciertas operaciones relacionadas con la consulta. Por ejemplo, puede ser utilizada para evitar que un método sea considerado en operaciones automáticas de persistencia o consulta dentro de un framework de almacenamiento.

Esta anotación no contiene ningún valor o parámetro adicional, actuando simplemente como una marca para el procesamiento en tiempo de ejecución.

Código Java

A continuación se presenta el código Java para la anotación `StorageName`:

```java
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();
}
```

Explicación Detallada del Código

La anotación `StorageName` se utiliza para asociar un nombre de almacenamiento a una clase. Esto puede ser útil en sistemas que requieren una asignación entre modelos de datos y sus correspondientes nombres en el almacenamiento.

. *Componentes de la Anotación*

- `@Target(ElementType.TYPE)`: Define que esta anotación puede aplicarse a tipos (es decir, a clases, interfaces, etc.).
- `@Retention(RetentionPolicy.RUNTIME)`: Indica que la anotación está disponible en tiempo de ejecución. Esto permite que la anotación sea accesible a través de reflexión en tiempo de ejecución.
- `String value()`: Es el único elemento de la anotación, que especifica el nombre de almacenamiento que se desea asociar con la clase anotada.

. *Uso*

La anotación `StorageName` se utiliza para proporcionar un nombre de almacenamiento que puede ser utilizado por frameworks o sistemas de persistencia para mapear una clase a una entidad en un sistema de almacenamiento, como una base de datos.

Por ejemplo, se podría usar la anotación para especificar el nombre de una tabla en una base de datos que corresponde a una clase en el código:

```
@StorageName("users")
public class User {
    // Campos y m todos de la clase
}
```

En este caso, la clase `User` está asociada con la tabla "users" en la base de datos.

V. Resultados

- La implementación de la clase Columns y sus subclases permite una gestión flexible y eficiente de columnas en modelos de datos. La capacidad de incluir o excluir columnas facilita la personalización de la representación de datos según las necesidades específicas de cada consulta o operación.
- La clase StorageException proporciona una forma estructurada de manejar errores relacionados con el almacenamiento. Al permitir mensajes personalizados y la inclusión de causas específicas, mejora la claridad y la gestión de excepciones en el sistema.
- La clase Hashing asegura que las contraseñas sean manejadas de manera segura mediante el uso de hashing y salting. Esto protege las contraseñas de los usuarios contra ataques de recuperación de contraseñas y asegura que el almacenamiento de contraseñas en la base de datos sea seguro.

VI. Discusión y Conclusiones

- La implementación de la clase Columns y sus subclases proporciona una solución flexible y extensible para la gestión de columnas en modelos de datos. La capacidad de incluir o excluir columnas según criterios específicos facilita la adaptación a diferentes necesidades de consulta y almacenamiento.
- La clase StorageException demuestra la importancia de contar con excepciones personalizadas para manejar errores específicos en sistemas de almacenamiento. Esta práctica mejora la capacidad de depuración y la claridad en el manejo de errores, permitiendo una mejor resolución de problemas.
- La combinación de la clase User y la clase Hashing ofrece un enfoque sólido para la gestión de datos de usuario y la seguridad de contraseñas. El uso de técnicas avanzadas de hashing y salting protege adecuadamente las contraseñas, lo que es esencial para mantener la integridad y seguridad del sistema.

Referencias

[1] K. Bella and A. Boulmakoul, "Containerised real-time architecture for safe urban navigation," in *2021 International Conference on Decision Aid Sciences and Application (DASA)*. IEEE, 2021, pp. 938–941.

[2] A. Boulmakoul, K. Bella, and A. Lbath, "Real-time distributed system for pedestrians' fuzzy safe navigation in urban environment," in *Intelligent and Fuzzy Techniques for Emerging Conditions and Digital Transformation: Proceedings of the INFUS 2021 Conference, held August 24-26, 2021. Volume 2*. Springer, 2022, pp. 655–662.

[3] N. Laranjeiro, J. Agnelo, and J. Bernardino, "A black box tool for robustness testing of rest services," *IEEE Access*, vol. 9, pp. 24 738–24 754, 2021.