Endpoint Server

1. Introducción

En este documento se va a documentar las necesidades que tiene el api server. Las dependencias ayudan con una gestión eficiente de datos y la implementación de consultas precisas son aspectos fundamentales en el desarrollo de aplicaciones robustas y escalables. Este documento explora varias implementaciones en Java relacionadas con la gestión de datos, incluyendo el almacenamiento en caché, el manejo de excepciones y la construcción de consultas. Se examinan clases como CacheManager, que se encarga de la administración de caché y el almacenamiento de posiciones de dispositivos; SmsManager, que maneja el envío de mensajes; y varias clases asociadas con la construcción de consultas (Columns, Condition, Request). Estas implementaciones son clave para garantizar la integridad de los datos, la seguridad y la eficiencia en las operaciones sobre bases de datos.

2. Objetivos

2.1. Objetivo General

Proporcionar una visión comprensiva sobre las clases y métodos utilizados en Java para la gestión de caché, el manejo de excepciones y la construcción de consultas.

2.2. Objetivos Específicos

- Detallar cómo estas clases se utilizan para construir consultas dinámicas y flexibles a bases de datos, y cómo facilitan la gestión de columnas y condiciones en las consultas.
- Revisar los diseños para el envío de mensajes y la gestión de excepciones, y cómo estas prácticas afectan la robustez del sistema de mensajería.
- Examinar la funcionalidad analizando cómo gestiona el almacenamiento en caché y la actualización de posiciones de dispositivos, evaluando su impacto en el rendimiento del sistema.

3. Desarrollo

```
package org.traccar.api.resource;
import org.traccar.api.BaseResource;
import org.traccar.model.ObjectOperation;
import org.traccar.config.Config;
import org.traccar.config.Keys;
import org.traccar.database.OpenIdProvider;
import org.traccar.geocoder.Geocoder;
import org.traccar.helper.Log;
import org.traccar.helper.LogAction;
import org.traccar.helper.model.UserUtil;
import org.traccar.mail.MailManager;
import org.traccar.model.Server;
import org.traccar.model.User;
import org.traccar.session.cache.CacheManager;
import org.traccar.sms.SmsManager;
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.annotation.Nullable;
import jakarta.annotation.security.PermitAll;
import jakarta.inject.Inject;
import jakarta.ws.rs.Consumes;
import jakarta.ws.rs.GET;
import jakarta.ws.rs.POST;
import jakarta.ws.rs.PUT;
import jakarta.ws.rs.Path;
```

```
import jakarta.ws.rs.PathParam;
import jakarta.ws.rs.Produces;
import jakarta.ws.rs.QueryParam;
import jakarta.ws.rs.core.MediaType;
import jakarta.ws.rs.core.Response;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.Arrays;
import java.util.Collection;
import java.util.TimeZone;

@Path("server")
@Produces(MediaType.APPLICATION_JSON)
@Consumes(MediaType.APPLICATION_JSON)
public class ServerResource extends BaseResource {

   @Inject
   private Config config;

   @Inject
   private CacheManager cacheManager;

   @Inject
   private MailManager mailManager;
```

```
@Inject
@Nullable
private SmsManager smsManager;

@Inject
@Nullable
private OpenIdProvider openIdProvider;

@Inject
@Nullable
private Geocoder geocoder;

@PermitAll
@GET
public Server get() throws StorageException {
    Server server = storage.getObject(Server.class, new Request(new Columns.All()));
    server.setEmailEnabled(mailManager.getEmailEnabled());
    server.setTextEnabled(smsManager != null);
    server.setGeocoderEnabled(geocoder != null);
    server.setOpenIdEnabled(openIdProvider != null);
    server.setOpenIdForce(openIdProvider != null && openIdProvider.getForce());
    User user = permissionsService.getUser(getUserId());
    if (user != null) {
        if (user.getAdministrator()) {
            server.setStorageSpace(Log.getStorageSpace());
        }
    } else {
        server.setNewServer(UserUtil.isEmpty(storage));
    }
```

```
        return server;
    }

    @PUT
    public Response update(Server server) throws Exception {
        permissionsService.checkAdmin(getUserId());
        storage.updateObject(server, new Request(
                new Columns.Exclude("id"),
                new Condition.Equals("id", server.getId())));
        cacheManager.invalidateObject(true, Server.class, server.getId(), ObjectOperation.UPDATE);
        LogAction.edit(getUserId(), server);
        return Response.ok(server).build();
    }

    @Path("geocode")
    @GET
    public String geocode(@QueryParam("latitude") double latitude, @QueryParam("longitude")
double longitude) {
        if (geocoder != null) {
            return geocoder.getAddress(latitude, longitude, null);
        } else {
            throw new RuntimeException("Reverse geocoding is not enabled");
        }
    }

    @Path("timezones")
    @GET
    public Collection<String> timezones() {
        return Arrays.asList(TimeZone.getAvailableIDs());
```

```
    }


    @Path("file/{path}")
    @POST
    @Consumes("*/*")
    public Response uploadFile(@PathParam("path") String path, File inputFile) throws IOException,
StorageException {
        permissionsService.checkAdmin(getUserId());
        String root = config.getString(Keys.WEB_OVERRIDE, config.getString(Keys.WEB_PATH));


        var rootPath = Paths.get(root).normalize();
        var outputPath = rootPath.resolve(path).normalize();
        if (!outputPath.startsWith(rootPath)) {
            return Response.status(Response.Status.BAD_REQUEST).build();
        }


        var directoryPath = outputPath.getParent();
        if (directoryPath != null) {
            Files.createDirectories(directoryPath);
        }


        try (var input = new FileInputStream(inputFile); var output = new
FileOutputStream(outputPath.toFile())) {
            input.transferTo(output);
        }
        return Response.ok().build();
    }


    @Path("cache")
```

```
    @GET
    public String cache() throws StorageException {
        permissionsService.checkAdmin(getUserId());
        return cacheManager.toString();
    }

    @Path("reboot")
    @POST
    public void reboot() throws StorageException {
        permissionsService.checkAdmin(getUserId());
        System.exit(130);
    }

}
```

3.1. Dependencias para la api server

- **import org.traccar.api.BaseResource;**

```
package org.traccar.api;
import org.traccar.api.security.PermissionsService;
import org.traccar.api.security.UserPrincipal;
import org.traccar.storage.Storage;
import jakarta.inject.Inject;
import jakarta.ws.rs.core.Context;
import jakarta.ws.rs.core.SecurityContext;

public class BaseResource {

    @Context
    private SecurityContext securityContext;

    @Inject
    protected Storage storage;

    @Inject
    protected PermissionsService permissionsService;

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

```
}
```

Dependencias:

- PermissionsService: Servicio que maneja permisos, probablemente usado para verificar si un usuario tiene ciertos derechos de acceso.
- UserPrincipal: Clase que representa al usuario autenticado, incluyendo su identificación.
- Storage: Maneja el almacenamiento, probablemente para interactuar con la base de datos.

Inyecciones de Dependencias (@Inject):

- **storage:** Proporciona acceso al sistema de almacenamiento (probablemente a la base de datos).
- **permissionsService:** Proporciona servicios relacionados con la verificación de permisos del usuario.

SecurityContext:

- **@Context SecurityContext:**
  El SecurityContext proporciona información de seguridad y autenticación del contexto actual de la solicitud. En este caso, se utiliza para obtener el principal del usuario autenticado.

Método getUserId:

- **getUserId()**:
  Este método obtiene la identificación del usuario actualmente autenticado. Utiliza el SecurityContext para obtener el principal (UserPrincipal) del usuario, y luego devuelve su userId. Si no hay un usuario autenticado (es decir, principal es null), el método devuelve 0.

BaseResource actúa como una superclase que proporciona funcionalidades comunes para las clases de recursos de la API en el sistema. Permite a las clases derivadas acceder fácilmente a los servicios de almacenamiento y permisos, y obtener la identificación del usuario autenticado. Esto facilita la reutilización de código y mantiene la lógica de autenticación y autorización centralizada.

- **import org.traccar.model.ObjectOperation;**

```
package org.traccar.model;


public enum ObjectOperation {
   ADD,
   UPDATE,
   DELETE,
}
```

ObjectOperation es una enumeración, lo que significa que define un conjunto de constantes, cada una representando un valor específico que puede tomar el tipo ObjectOperation. En este caso, las constantes son:

- **ADD:** Representa la operación de agregar un nuevo objeto al sistema.
- **UPDATE:** Representa la operación de actualizar un objeto existente.
- **DELETE:** Representa la operación de eliminar un objeto del sistema.

- **import org.traccar.config.Config;**

```
package org.traccar.config;


import com.google.common.annotations.VisibleForTesting;
import com.google.inject.name.Named;
import org.traccar.helper.Log;

import jakarta.inject.Inject;
import jakarta.inject.Singleton;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.util.InvalidPropertiesFormatException;
import java.util.Objects;
import java.util.Properties;

@Singleton
public class Config {

    private final Properties properties = new Properties();

    private boolean useEnvironmentVariables;

    public Config() {
    }

    @Inject
    public Config(@Named("configFile") String file) throws IOException {
        try {
            try (InputStream inputStream = new FileInputStream(file)) {
                properties.loadFromXML(inputStream);
```

```
            }

            useEnvironmentVariables =
Boolean.parseBoolean(System.getenv("CONFIG_USE_ENVIRONMENT_VARIABLES"))
                    || Boolean.parseBoolean(properties.getProperty("config.useEnvironmentVariables"));

            Log.setupLogger(this);
        } catch (InvalidPropertiesFormatException e) {
            Log.setupDefaultLogger();
            throw new RuntimeException("Configuration file is not a valid XML document", e);
        } catch (Exception e) {
            Log.setupDefaultLogger();
            throw e;
        }
    }

    public boolean hasKey(ConfigKey<?> key) {
        return hasKey(key.getKey());
    }

    private boolean hasKey(String key) {
        return useEnvironmentVariables &&
System.getenv().containsKey(getEnvironmentVariableName(key))
                || properties.containsKey(key);
    }

    public String getString(ConfigKey<String> key) {
        return getString(key.getKey(), key.getDefaultValue());
    }

    @Deprecated
    public String getString(String key) {
        if (useEnvironmentVariables) {
            String value = System.getenv(getEnvironmentVariableName(key));
            if (value != null && !value.isEmpty()) {
                return value;
            }
        }
        return properties.getProperty(key);
    }

    public String getString(ConfigKey<String> key, String defaultValue) {
        return getString(key.getKey(), defaultValue);
    }
```

```
@Deprecated
public String getString(String key, String defaultValue) {
    return hasKey(key) ? getString(key) : defaultValue;
}

public boolean getBoolean(ConfigKey<Boolean> key) {
    String value = getString(key.getKey());
    if (value != null) {
        return Boolean.parseBoolean(value);
    } else {
        Boolean defaultValue = key.getDefaultValue();
        return Objects.requireNonNullElse(defaultValue, false);
    }
}

public int getInteger(ConfigKey<Integer> key) {
    String value = getString(key.getKey());
    if (value != null) {
        return Integer.parseInt(value);
    } else {
        Integer defaultValue = key.getDefaultValue();
        return Objects.requireNonNullElse(defaultValue, 0);
    }
}

public int getInteger(ConfigKey<Integer> key, int defaultValue) {
    return getInteger(key.getKey(), defaultValue);
}

@Deprecated
public int getInteger(String key, int defaultValue) {
    return hasKey(key) ? Integer.parseInt(getString(key)) : defaultValue;
}

public long getLong(ConfigKey<Long> key) {
    String value = getString(key.getKey());
    if (value != null) {
        return Long.parseLong(value);
    } else {
        Long defaultValue = key.getDefaultValue();
        return Objects.requireNonNullElse(defaultValue, 0L);
    }
}

public double getDouble(ConfigKey<Double> key) {
```

```
        String value = getString(key.getKey());
        if (value != null) {
            return Double.parseDouble(value);
        } else {
            Double defaultValue = key.getDefaultValue();
            return Objects.requireNonNullElse(defaultValue, 0.0);
        }
    }

    @VisibleForTesting
    public void setString(ConfigKey<?> key, String value) {
        properties.put(key.getKey(), value);
    }

    static String getEnvironmentVariableName(String key) {
        return key.replaceAll("\\.", "_").replaceAll("(\\p{Lu})", "_$1").toUpperCase();
    }

}
```

- **Cargar Configuración desde un Archivo XML:**

  La clase Config tiene un constructor que permite cargar las configuraciones desde un archivo XML. Utiliza un FileInputStream para leer el archivo, y el método properties.loadFromXML para cargar las propiedades definidas en ese archivo.

  Si el archivo de configuración no es un documento XML válido, se lanza una excepción InvalidPropertiesFormatException.

- **Uso de Variables de Entorno:**

  La clase permite configurar si se deben utilizar variables de entorno para sobrescribir las propiedades cargadas desde el archivo XML. Esto se controla a través de la variable de entorno CONFIG_USE_ENVIRONMENT_VARIABLES o la propiedad config.useEnvironmentVariables.

- **Métodos para Obtener Valores de Configuración:**

  La clase proporciona varios métodos para obtener valores de configuración de diferentes tipos: String, boolean, int, long, y double. Estos métodos permiten obtener

el valor de una clave específica, utilizando las propiedades cargadas o las variables de entorno. Si no se encuentra una clave, se pueden proporcionar valores por defecto.

- **Registrar Valores para Pruebas:**

  El método setString está anotado con @VisibleForTesting, lo que indica que está pensado principalmente para ser usado en pruebas, permitiendo sobrescribir o registrar valores en las propiedades.

- **Uso de ConfigKey:**

  La clase usa un objeto ConfigKey para manejar las claves de configuración, lo que proporciona un manejo más tipado de las propiedades.

- **import org.traccar.database.OpenIdProvider;**

```
package org.traccar.database;


import org.traccar.config.Config;
import org.traccar.config.Keys;
import org.traccar.api.resource.SessionResource;
import org.traccar.api.security.LoginService;
import org.traccar.model.User;
import org.traccar.storage.StorageException;
import org.traccar.helper.LogAction;
import org.traccar.helper.WebHelper;

import java.net.URI;
import java.net.URISyntaxException;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse.BodyHandlers;
import java.security.GeneralSecurityException;
import java.util.List;
import java.util.Map;
import java.io.IOException;
import jakarta.servlet.http.HttpServletRequest;

import com.fasterxml.jackson.core.type.TypeReference;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.google.inject.Inject;

import com.nimbusds.oauth2.sdk.http.HTTPResponse;
import com.nimbusds.oauth2.sdk.AuthorizationCode;
```

```
import com.nimbusds.oauth2.sdk.ResponseType;
import com.nimbusds.oauth2.sdk.Scope;
import com.nimbusds.oauth2.sdk.AuthorizationGrant;
import com.nimbusds.oauth2.sdk.TokenRequest;
import com.nimbusds.oauth2.sdk.TokenResponse;
import com.nimbusds.oauth2.sdk.AuthorizationCodeGrant;
import com.nimbusds.oauth2.sdk.ParseException;
import com.nimbusds.oauth2.sdk.AuthorizationResponse;
import com.nimbusds.oauth2.sdk.auth.Secret;
import com.nimbusds.oauth2.sdk.auth.ClientSecretBasic;
import com.nimbusds.oauth2.sdk.auth.ClientAuthentication;
import com.nimbusds.oauth2.sdk.token.BearerAccessToken;
import com.nimbusds.oauth2.sdk.id.State;
import com.nimbusds.oauth2.sdk.id.ClientID;
import com.nimbusds.openid.connect.sdk.OIDCTokenResponse;
import com.nimbusds.openid.connect.sdk.OIDCTokenResponseParser;
import com.nimbusds.openid.connect.sdk.UserInfoResponse;
import com.nimbusds.openid.connect.sdk.UserInfoRequest;
import com.nimbusds.openid.connect.sdk.AuthenticationRequest;
import com.nimbusds.openid.connect.sdk.claims.UserInfo;

public class OpenIdProvider {
    private final Boolean force;
    private final ClientID clientId;
    private final ClientAuthentication clientAuth;
    private final URI callbackUrl;
    private final URI authUrl;
    private final URI tokenUrl;
    private final URI userInfoUrl;
    private final URI baseUrl;
    private final String adminGroup;
    private final String allowGroup;

    private final LoginService loginService;

    @Inject
    public OpenIdProvider(Config config, LoginService loginService, HttpClient httpClient,
ObjectMapper objectMapper)
        throws InterruptedException, IOException, URISyntaxException {

        this.loginService = loginService;

        force = config.getBoolean(Keys.OPENID_FORCE);
        clientId = new ClientID(config.getString(Keys.OPENID_CLIENT_ID));
```

```
        clientAuth = new ClientSecretBasic(clientId, new
Secret(config.getString(Keys.OPENID_CLIENT_SECRET)));

        baseUrl = new URI(WebHelper.retrieveWebUrl(config));
        callbackUrl = new URI(WebHelper.retrieveWebUrl(config) + "/api/session/openid/callback");

        if (config.hasKey(Keys.OPENID_ISSUER_URL)) {
            HttpRequest httpRequest = HttpRequest.newBuilder(
                URI.create(config.getString(Keys.OPENID_ISSUER_URL) + "/.well-known/openid-
configuration"))
                .header("Accept", "application/json")
                .build();

            String httpResponse = httpClient.send(httpRequest, BodyHandlers.ofString()).body();

            Map<String, Object> discoveryMap = objectMapper.readValue(httpResponse, new
TypeReference<>() {
            });

            authUrl = new URI((String) discoveryMap.get("authorization_endpoint"));
            tokenUrl = new URI((String) discoveryMap.get("token_endpoint"));
            userInfoUrl = new URI((String) discoveryMap.get("userinfo_endpoint"));
        } else {
            authUrl = new URI(config.getString(Keys.OPENID_AUTH_URL));
            tokenUrl = new URI(config.getString(Keys.OPENID_TOKEN_URL));
            userInfoUrl = new URI(config.getString(Keys.OPENID_USERINFO_URL));
        }

        adminGroup = config.getString(Keys.OPENID_ADMIN_GROUP);
        allowGroup = config.getString(Keys.OPENID_ALLOW_GROUP);
    }

    public URI createAuthUri() {
        Scope scope = new Scope("openid", "profile", "email");

        if (adminGroup != null) {
            scope.add("groups");
        }

        AuthenticationRequest.Builder request = new AuthenticationRequest.Builder(
                new ResponseType("code"),
                scope,
                clientId,
                callbackUrl);
```

```
        return request.endpointURI(authUrl)
                .state(new State())
                .build()
                .toURI();
    }

    private OIDCTokenResponse getToken(AuthorizationCode code)
            throws IOException, ParseException, GeneralSecurityException {
        AuthorizationGrant codeGrant = new AuthorizationCodeGrant(code, callbackUrl);
        TokenRequest tokenRequest = new TokenRequest(tokenUrl, clientAuth, codeGrant);

        HTTPResponse tokenResponse = tokenRequest.toHTTPRequest().send();
        TokenResponse token = OIDCTokenResponseParser.parse(tokenResponse);
        if (!token.indicatesSuccess()) {
            throw new GeneralSecurityException("Unable to authenticate with the OpenID Connect
provider.");
        }

        return (OIDCTokenResponse) token.toSuccessResponse();
    }

    private UserInfo getUserInfo(BearerAccessToken token) throws IOException, ParseException,
GeneralSecurityException {
        HTTPResponse httpResponse = new UserInfoRequest(userInfoUrl, token)
                .toHTTPRequest()
                .send();

        UserInfoResponse userInfoResponse = UserInfoResponse.parse(httpResponse);

        if (!userInfoResponse.indicatesSuccess()) {
            throw new GeneralSecurityException(
                "Failed to access OpenID Connect user info endpoint. Please contact your administrator.");
        }

        return userInfoResponse.toSuccessResponse().getUserInfo();
    }

    public URI handleCallback(URI requestUri, HttpServletRequest request)
            throws StorageException, ParseException, IOException, GeneralSecurityException {

        AuthorizationResponse response = AuthorizationResponse.parse(requestUri);

        if (!response.indicatesSuccess()) {
            throw new
GeneralSecurityException(response.toErrorResponse().getErrorObject().getDescription());
```

```
        }

        AuthorizationCode authCode = response.toSuccessResponse().getAuthorizationCode();

        if (authCode == null) {
            throw new GeneralSecurityException("Malformed OpenID callback.");
        }

        OIDCTokenResponse tokens = getToken(authCode);

        BearerAccessToken bearerToken = tokens.getOIDCTokens().getBearerAccessToken();

        UserInfo userInfo = getUserInfo(bearerToken);

        List<String> userGroups = userInfo.getStringListClaim("groups");
        boolean administrator = adminGroup != null && userGroups.contains(adminGroup);

        if (!(administrator || allowGroup == null || userGroups.contains(allowGroup))) {
            throw new GeneralSecurityException("Your OpenID Groups do not permit access to Traccar.");
        }

        User user = loginService.login(
                userInfo.getEmailAddress(), userInfo.getName(), administrator).getUser();

        request.getSession().setAttribute(SessionResource.USER_ID_KEY, user.getId());
        LogAction.login(user.getId(), WebHelper.retrieveRemoteAddress(request));

        return baseUrl;
    }

    public boolean getForce() {
        return force;
    }
}
```

Atributos de Configuración:

- La clase define varios atributos clave para manejar la autenticación, como clientId, clientAuth, callbackUrl, authUrl, tokenUrl, userInfoUrl, y baseUrl, que se inicializan usando la clase Config. Estos parámetros son necesarios para interactuar con el proveedor OpenID Connect.
- Además, existen atributos como adminGroup y allowGroup para manejar los permisos de usuario basados en grupos.

Constructor:

- El constructor de la clase recibe como parámetros una instancia de Config, un LoginService, un HttpClient y un ObjectMapper.
- Se configura la autenticación usando las propiedades almacenadas en config. Si se proporciona una URL de emisor (issuer), se realiza una solicitud HTTP para obtener la configuración OpenID Connect usando la URL /.well-known/openid-configuration.

Creación de la URI de Autenticación:

- El método createAuthUri construye y retorna una URI de autenticación que se utiliza para redirigir a los usuarios al proveedor OpenID Connect. Esta URI incluye el scope requerido, como openid, profile, y email, además de la clientId y callbackUrl.

Intercambio de Código de Autorización por Tokens:

- El método privado getToken se encarga de intercambiar el código de autorización (AuthorizationCode) recibido en la respuesta de autenticación por un token de acceso (BearerAccessToken) mediante una solicitud a la URL de token del proveedor OpenID Connect.

Obtención de Información del Usuario:

- El método privado getUserInfo utiliza el token de acceso para realizar una solicitud a la URL de userinfo y recuperar la información del usuario autenticado. Si la solicitud falla, se lanza una excepción de seguridad.

Manejo de la Respuesta del Callback:

- El método handleCallback es invocado después de que el proveedor OpenID Connect redirige al usuario de vuelta a la aplicación. Este método valida la respuesta de autenticación, intercambia el código de autorización por tokens, obtiene la información del usuario, y finalmente inicia la sesión del usuario en la aplicación.
- También se verifica si el usuario pertenece a los grupos permitidos (adminGroup o allowGroup) antes de conceder el acceso.

Método para Forzar Autenticación:

- El método getForce simplemente retorna el valor de la propiedad force, que indica si la autenticación OpenID debe ser forzada.

- **import org.traccar.geocoder.Geocoder;**

```java
package org.traccar.geocoder;

import org.traccar.database.StatisticsManager;

public interface Geocoder {

    interface ReverseGeocoderCallback {

        void onSuccess(String address);

        void onFailure(Throwable e);

    }

    String getAddress(double latitude, double longitude, ReverseGeocoderCallback
callback);

    void setStatisticsManager(StatisticsManager statisticsManager);

}
```

Método para Obtener una Dirección:

- El método principal de esta interfaz es getAddress, que toma como parámetros una latitud y una longitud, además de un callback (ReverseGeocoderCallback).
- Este método se utiliza para realizar la geocodificación inversa, es decir, convertir coordenadas geográficas (latitud y longitud) en una dirección de texto.
- El callback permite manejar de forma asíncrona el resultado de la operación, notificando si la operación fue exitosa (onSuccess) o si ocurrió un error (onFailure).

Interfaz ReverseGeocoderCallback:

- Esta interfaz interna define dos métodos:
    - onSuccess(String address): Se invoca cuando la operación de geocodificación inversa es exitosa, proporcionando la dirección resultante.
    - onFailure(Throwable e): Se invoca cuando la operación falla, pasando la excepción que ocurrió como parámetro para manejar el error.

Método para Configurar el StatisticsManager:

- El método setStatisticsManager permite configurar un StatisticsManager, que es responsable de gestionar estadísticas relacionadas con las operaciones de geocodificación.

- Registrar el número de solicitudes, tiempos de respuesta, tasas de éxito, entre otros indicadores de rendimiento.

- **import org.traccar.helper.Log;**

```
package org.traccar.helper;


import org.traccar.config.Config;
import org.traccar.config.Keys;
import org.traccar.model.Pair;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.io.StringWriter;
import java.io.Writer;
import java.net.URL;
import java.nio.charset.StandardCharsets;
import java.nio.file.FileStore;
import java.nio.file.FileSystems;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.Date;
import java.util.logging.ConsoleHandler;
import java.util.logging.Formatter;
import java.util.logging.Handler;
import java.util.logging.Level;
import java.util.logging.LogRecord;
import java.util.logging.Logger;
import java.util.stream.Stream;

public final class Log {

    private Log() {
    }

    private static final String STACK_PACKAGE = "org.traccar";
    private static final int STACK_LIMIT = 4;

    private static class RollingFileHandler extends Handler {
```

```
    private final String name;
    private String suffix;
    private Writer writer;
    private final boolean rotate;
    private final String template;

    RollingFileHandler(String name, boolean rotate, String rotateInterval) {
        this.name = name;
        this.rotate = rotate;
        this.template = rotateInterval.equalsIgnoreCase("HOUR") ? "yyyyMMddHH" :
"yyyyMMdd";
    }

    @Override
    public synchronized void publish(LogRecord record) {
        if (isLoggable(record)) {
            try {
                String suffix = "";
                if (rotate) {
                    suffix = new SimpleDateFormat(template).format(new
Date(record.getMillis()));
                    if (writer != null && !suffix.equals(this.suffix)) {
                        writer.close();
                        writer = null;
                        if (!new File(name).renameTo(new File(name + "." + this.suffix))) {
                            throw new RuntimeException("Log file renaming failed");
                        }
                    }
                }
                if (writer == null) {
                    this.suffix = suffix;
                    writer = new BufferedWriter(
                            new OutputStreamWriter(new FileOutputStream(name, true),
StandardCharsets.UTF_8));
                }
                writer.write(getFormatter().format(record));
                writer.flush();
            } catch (IOException e) {
                throw new RuntimeException(e);
            }
        }
    }

    @Override
    public synchronized void flush() {
```

```
        if (writer != null) {
            try {
                writer.flush();
            } catch (IOException e) {
                throw new RuntimeException(e);
            }
        }
    }

    @Override
    public synchronized void close() throws SecurityException {
        if (writer != null) {
            try {
                writer.close();
            } catch (IOException e) {
                throw new RuntimeException(e);
            }
        }
    }

}

public static class LogFormatter extends Formatter {

    private final boolean fullStackTraces;

    LogFormatter(boolean fullStackTraces) {
        this.fullStackTraces = fullStackTraces;
    }

    private static String formatLevel(Level level) {
        return switch (level.getName()) {
            case "FINEST" -> "TRACE";
            case "FINER", "FINE", "CONFIG" -> "DEBUG";
            case "INFO" -> "INFO";
            case "WARNING" -> "WARN";
            default -> "ERROR";
        };
    }

    @Override
    public String format(LogRecord record) {
        StringBuilder message = new StringBuilder();

        if (record.getMessage() != null) {
```

```
            message.append(record.getMessage());
        }

        if (record.getThrown() != null) {
            if (!message.isEmpty()) {
                message.append(" - ");
            }
            if (fullStackTraces) {
                StringWriter stringWriter = new StringWriter();
                PrintWriter printWriter = new PrintWriter(stringWriter);
                record.getThrown().printStackTrace(printWriter);
                message.append(System.lineSeparator()).append(stringWriter.toString());
            } else {
                message.append(exceptionStack(record.getThrown()));
            }
        }

        return String.format("%1$tF %1$tT %2$5s: %3$s%n",
                new Date(record.getMillis()), formatLevel(record.getLevel()),
message.toString());
    }

  }

  public static void setupDefaultLogger() {
    String path = null;
    URL url = ClassLoader.getSystemClassLoader().getResource(".");
    if (url != null) {
      File jarPath = new File(url.getPath());
      File logsPath = new File(jarPath, "logs");
      if (!logsPath.exists() || !logsPath.isDirectory()) {
        logsPath = jarPath;
      }
      path = new File(logsPath, "tracker-server.log").getPath();
    }
    setupLogger(path == null, path, Level.WARNING.getName(), false, true, "DAY");
  }

  public static void setupLogger(Config config) {
    setupLogger(
        config.getBoolean(Keys.LOGGER_CONSOLE),
        config.getString(Keys.LOGGER_FILE),
        config.getString(Keys.LOGGER_LEVEL),
        config.getBoolean(Keys.LOGGER_FULL_STACK_TRACES),
        config.getBoolean(Keys.LOGGER_ROTATE),
```

```
            config.getString(Keys.LOGGER_ROTATE_INTERVAL));
    }

    private static void setupLogger(
            boolean console, String file, String levelString,
            boolean fullStackTraces, boolean rotate, String rotateInterval) {

        Logger rootLogger = Logger.getLogger("");
        for (Handler handler : rootLogger.getHandlers()) {
            rootLogger.removeHandler(handler);
        }

        Handler handler;
        if (console) {
            handler = new ConsoleHandler();
        } else {
            handler = new RollingFileHandler(file, rotate, rotateInterval);
        }

        handler.setFormatter(new LogFormatter(fullStackTraces));

        Level level = Level.parse(levelString.toUpperCase());
        rootLogger.setLevel(level);
        handler.setLevel(level);
        handler.setFilter(record -> record != null &&
!record.getLoggerName().startsWith("sun"));

        rootLogger.addHandler(handler);
    }

    public static String exceptionStack(Throwable exception) {
        Throwable cause = exception.getCause();
        while (cause != null && exception != cause) {
            exception = cause;
            cause = cause.getCause();
        }

        StringBuilder s = new StringBuilder();
        String exceptionMsg = exception.getMessage();
        if (exceptionMsg != null) {
            s.append(exceptionMsg);
            s.append(" - ");
        }
        s.append(exception.getClass().getSimpleName());
        StackTraceElement[] stack = exception.getStackTrace();
```

```
    if (stack.length > 0) {
        int count = STACK_LIMIT;
        boolean first = true;
        boolean skip = false;
        String file = "";
        s.append(" (");
        for (StackTraceElement element : stack) {
            if (count > 0 && element.getClassName().startsWith(STACK_PACKAGE)) {
                if (!first) {
                    s.append(" < ");
                } else {
                    first = false;
                }

                if (skip) {
                    s.append("... < ");
                    skip = false;
                }

                if (file.equals(element.getFileName())) {
                    s.append("*");
                } else {
                    file = element.getFileName();
                    s.append(file, 0, file.length() - 5); // remove ".java"
                    count -= 1;
                }
                s.append(":").append(element.getLineNumber());
            } else {
                skip = true;
            }
        }
        if (skip) {
            if (!first) {
                s.append(" < ");
            }
            s.append("...");
        }
        s.append(")");
    }
    return s.toString();
}

public static long[] getStorageSpace() {
    var stores = new ArrayList<Pair<Long, Long>>();
```

```
        for (FileStore store : FileSystems.getDefault().getFileStores()) {
            try {
                long usableSpace = store.getUsableSpace();
                long totalSpace = store.getTotalSpace();
                if (totalSpace > 1_000_000_000) {
                    stores.add(new Pair<>(usableSpace, totalSpace));
                }
            } catch (IOException ignored) {
            }
        }
        return stores.stream()
                .sorted(Comparator.comparingDouble(p -> p.first() / (double) p.second()))
                .flatMap(p -> Stream.of(p.first(), p.second()))
                .mapToLong(Long::longValue)
                .toArray();
    }

}
```

Proporciona un sistema de logging con soporte para rotación de archivos, formateo de logs y manejo de excepciones, lo cual es esencial para la monitorización y depuración de aplicaciones en producción. La clase también incluye herramientas para monitorear el almacenamiento, asegurando que el sistema pueda gestionar eficientemente los recursos mientras mantiene un registro detallado de las operaciones.

Gestión de Logs: La clase Log centraliza la configuración y el manejo de logs en la aplicación. Es capaz de formatear logs, manejar excepciones, y registrar información en archivos de log que rotan automáticamente.

Rotación de Archivos: La funcionalidad de rotación permite que los archivos de log sean renombrados y reiniciados en función de un intervalo de tiempo (por ejemplo, cada hora o día), lo que ayuda a gestionar el tamaño de los archivos de log.

Registro de Excepciones: La clase tiene métodos especializados para capturar y registrar excepciones de manera eficiente, incluyendo un resumen de la traza de la pila.

Monitoreo de Espacio de Almacenamiento: También incluye una funcionalidad para monitorear el espacio de almacenamiento disponible, lo que podría ser útil para asegurarse de que los logs no llenen el disco.

- **import org.traccar.helper.LogAction;**

```java
import java.beans.Introspector;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.List;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.traccar.model.BaseModel;

public final class LogAction {

    private static final Logger LOGGER =
LoggerFactory.getLogger(LogAction.class);

    private LogAction() {
    }

    private static final String ACTION_CREATE = "create";
    private static final String ACTION_EDIT = "edit";
    private static final String ACTION_REMOVE = "remove";

    private static final String ACTION_LINK = "link";
    private static final String ACTION_UNLINK = "unlink";

    private static final String ACTION_LOGIN = "login";
    private static final String ACTION_LOGOUT = "logout";

    private static final String ACTION_ACCUMULATORS = "accumulators";
    private static final String ACTION_COMMAND = "command";

    private static final String PATTERN_OBJECT = "user: %d, action: %s, object:
%s, id: %d";
    private static final String PATTERN_LINK = "user: %d, action: %s, owner: %s,
id: %d, property: %s, id: %d";
    private static final String PATTERN_LOGIN = "user: %d, action: %s, from: %s";
    private static final String PATTERN_LOGIN_FAILED = "login failed from: %s";
    private static final String PATTERN_ACCUMULATORS = "user: %d, action:
%s, deviceId: %d";
    private static final String PATTERN_COMMAND_DEVICE = "user: %d, action:
%s, deviceId: %d, type: %s";
    private static final String PATTERN_COMMAND_GROUP = "user: %d, action:
%s, groupId: %d, type: %s";
```

```
    private static final String PATTERN_REPORT = "user: %d, %s: %s, from: %s, to:
%s, devices: %s, groups: %s";

    public static void create(long userId, BaseModel object) {
        logObjectAction(ACTION_CREATE, userId, object.getClass(), object.getId());
    }

    public static void edit(long userId, BaseModel object) {
        logObjectAction(ACTION_EDIT, userId, object.getClass(), object.getId());
    }

    public static void remove(long userId, Class<?> clazz, long objectId) {
        logObjectAction(ACTION_REMOVE, userId, clazz, objectId);
    }

    public static void link(long userId, Class<?> owner, long ownerId, Class<?>
property, long propertyId) {
        logLinkAction(ACTION_LINK, userId, owner, ownerId, property, propertyId);
    }

    public static void unlink(long userId, Class<?> owner, long ownerId, Class<?>
property, long propertyId) {
        logLinkAction(ACTION_UNLINK, userId, owner, ownerId, property,
propertyId);
    }

    public static void login(long userId, String remoteAddress) {
        logLoginAction(ACTION_LOGIN, userId, remoteAddress);
    }

    public static void logout(long userId, String remoteAddress) {
        logLoginAction(ACTION_LOGOUT, userId, remoteAddress);
    }

    public static void failedLogin(String remoteAddress) {
        if (remoteAddress == null || remoteAddress.isEmpty()) {
            remoteAddress = "unknown";
        }
        LOGGER.info(String.format(PATTERN_LOGIN_FAILED, remoteAddress));
    }

    public static void resetAccumulators(long userId, long deviceId) {
        LOGGER.info(String.format(
                PATTERN_ACCUMULATORS, userId, ACTION_ACCUMULATORS,
deviceId));
```

```
    }

    public static void command(long userId, long groupId, long deviceId, String type)
{
        if (groupId > 0) {
            LOGGER.info(String.format(PATTERN_COMMAND_GROUP, userId,
ACTION_COMMAND, groupId, type));
        } else {
            LOGGER.info(String.format(PATTERN_COMMAND_DEVICE, userId,
ACTION_COMMAND, deviceId, type));
        }
    }

    public static void report(
            long userId, boolean scheduled, String report,
            Date from, Date to, List<Long> deviceIds, List<Long> groupIds) {
        DateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm");
        LOGGER.info(String.format(
                PATTERN_REPORT, userId, scheduled ? "scheduled" : "report", report,
                dateFormat.format(from), dateFormat.format(to),
                deviceIds.toString(), groupIds.toString()));
    }

    private static void logObjectAction(String action, long userId, Class<?> clazz,
long objectId) {
        LOGGER.info(String.format(
                PATTERN_OBJECT, userId, action,
Introspector.decapitalize(clazz.getSimpleName()), objectId));
    }

    private static void logLinkAction(
            String action, long userId, Class<?> owner, long ownerId, Class<?> property,
long propertyId) {
        LOGGER.info(String.format(
                PATTERN_LINK, userId, action,
                Introspector.decapitalize(owner.getSimpleName()), ownerId,
                Introspector.decapitalize(property.getSimpleName()), propertyId));
    }

    private static void logLoginAction(String action, long userId, String
remoteAddress) {
        if (remoteAddress == null || remoteAddress.isEmpty()) {
            remoteAddress = "unknown";
        }
```

```
        LOGGER.info(String.format(PATTERN_LOGIN, userId, action,
remoteAddress));
    }

}
```

LogAction es una solución bien estructurada para el registro de acciones dentro de una aplicación, proporcionando una API consistente y flexible para el logging. Su diseño permite un registro detallado y organizado de eventos, lo que es crucial para la monitorización, auditoría, y depuración en aplicaciones en producción.

Registro de Acciones: LogAction centraliza el registro de acciones dentro de la aplicación, permitiendo que las actividades clave como la manipulación de objetos, el login, y la generación de reportes sean registradas de manera consistente.

Flexibilidad: Al utilizar SLF4J, la clase se mantiene independiente de una implementación específica de logging, lo que permite cambiar el backend de logging (como Logback o Log4j2) sin modificar el código.

Consistencia en los Logs: Al definir patrones de log comunes y métodos reutilizables para registrar diferentes tipos de eventos, la clase asegura que los logs sean consistentes y fáciles de leer.

Manejo de Excepciones y Fallos: Aunque no se menciona explícitamente el manejo de excepciones en los métodos, la clase garantiza que, por ejemplo, en un fallo de login, la acción sea registrada con un mensaje específico, facilitando la depuración.

- **import org.traccar.helper.model.UserUtil;**

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

import org.traccar.config.Keys;
import org.traccar.model.Server;
import org.traccar.model.User;
import org.traccar.storage.Storage;
import org.traccar.storage.StorageException;
import org.traccar.storage.query.Columns;
import org.traccar.storage.query.Order;
import org.traccar.storage.query.Request;

import java.util.Date;
import java.util.TimeZone;
```

```
public final class UserUtil {

    private UserUtil() {
    }

    public static boolean isEmpty(Storage storage) throws StorageException {
        return storage.getObjects(User.class, new Request(
                new Columns.Include("id"),
                new Order("id", false, 1))).isEmpty();
    }

    public static String getDistanceUnit(Server server, User user) {
        return lookupStringAttribute(server, user, "distanceUnit", "km");
    }

    public static String getSpeedUnit(Server server, User user) {
        return lookupStringAttribute(server, user, "speedUnit", "kn");
    }

    public static String getVolumeUnit(Server server, User user) {
        return lookupStringAttribute(server, user, "volumeUnit", "ltr");
    }

    public static TimeZone getTimezone(Server server, User user) {
        String timezone = lookupStringAttribute(server, user, "timezone", null);
        return timezone != null ? TimeZone.getTimeZone(timezone) :
TimeZone.getDefault();
    }

    private static String lookupStringAttribute(Server server, User user, String key, String
defaultValue) {
        String preference;
        String serverPreference = server.getString(key);
        String userPreference = user.getString(key);
        if (server.getForceSettings()) {
            preference = serverPreference != null ? serverPreference : userPreference;
        } else {
            preference = userPreference != null ? userPreference : serverPreference;
        }
        return preference != null ? preference : defaultValue;
    }

    public static void setUserDefaults(User user, Config config) {
```

```
        user.setDeviceLimit(config.getInteger(Keys.USERS_DEFAULT_DEVICE_LIMIT)
);
        int expirationDays =
config.getInteger(Keys.USERS_DEFAULT_EXPIRATION_DAYS);
        if (expirationDays > 0) {
            user.setExpirationTime(new Date(System.currentTimeMillis() + expirationDays
* 86400000L));
        }
    }
}
```

UserUtil proporciona una serie de utilidades esenciales para la gestión de usuarios dentro de la aplicación, facilitando la obtención de configuraciones personalizadas y la inicialización de nuevos usuarios. Su diseño asegura flexibilidad y consistencia en la manera en que se aplican las configuraciones, tanto a nivel de usuario como de servidor.

Gestión de Preferencias de Usuario: UserUtil centraliza la lógica de obtener preferencias del usuario, como unidades de medida y zona horaria, teniendo en cuenta tanto las preferencias del usuario como las configuraciones del servidor. Esto asegura que las preferencias del usuario se manejen de manera consistente y que las configuraciones forzadas del servidor se respeten cuando sea necesario.

Inicialización de Usuarios Nuevos: setUserDefaults facilita la inicialización de nuevos usuarios con valores predeterminados, asegurando que todos los usuarios comiencen con una configuración básica consistente.

Consulta de Existencia de Usuarios: El método isEmpty es útil para determinar si se debe realizar alguna acción particular cuando no hay usuarios en el sistema, como configurar un administrador inicial.

- **import org.traccar.mail.MailManager;**

```
package org.traccar.mail;


import org.traccar.model.User;

import jakarta.mail.MessagingException;
import jakarta.mail.internet.MimeBodyPart;

public interface MailManager {
```

```
    boolean getEmailEnabled();

    void sendMessage(
            User user, boolean system, String subject, String body) throws
MessagingException;

    void sendMessage(
            User user, boolean system, String subject, String body, MimeBodyPart
attachment) throws MessagingException;

}
```

MailManager es una interfaz crítica para la funcionalidad de envío de correos electrónicos en una aplicación, asegurando que cualquier clase que la implemente pueda gestionar el envío de correos, con o sin archivos adjuntos, y verificar si el sistema de correo electrónico está habilitado. Esto es útil en escenarios donde la aplicación necesita comunicarse con los usuarios por medio de correos electrónicos, como notificaciones, alertas, o informes.

La interfaz MailManager proporciona la estructura básica para cualquier gestor de correos electrónicos dentro de la aplicación. Define métodos esenciales para verificar la disponibilidad del sistema de correo electrónico y para enviar mensajes tanto simples como con archivos adjuntos.

- **import org.traccar.model.User;**

```
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;
```

```java
@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);
    }

}
```

La clase User en org.traccar.model es una representación detallada de un usuario dentro del sistema Traccar. Proporciona métodos y atributos para manejar la autenticación segura, la configuración de la cuenta, y la aplicación de restricciones y permisos. Utiliza anotaciones para gestionar la persistencia y serialización de manera eficiente, asegurando que la información sensible esté protegida. Al implementar las interfaces UserRestrictions y Disableable, la clase asegura que las restricciones y la habilitación/deshabilitación de usuarios estén correctamente gestionadas en el sistema.

- **import org.traccar.session.cache.CacheManager;**

```
import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;
import java.util.concurrent.locks.ReadWriteLock;
import java.util.concurrent.locks.ReentrantReadWriteLock;
import java.util.stream.Collectors;

@Singleton
public class CacheManager implements BroadcastInterface {
```

```
    private static final Logger LOGGER =
LoggerFactory.getLogger(CacheManager.class);

    private static final Set<Class<? extends BaseModel>> GROUPED_CLASSES =
            Set.of(Attribute.class, Driver.class, Geofence.class,
Maintenance.class, Notification.class);

    private final Config config;
    private final Storage storage;
    private final BroadcastService broadcastService;

    private final ReadWriteLock lock = new ReentrantReadWriteLock();

    private final CacheGraph graph = new CacheGraph();

    private Server server;
    private final Map<Long, Position> devicePositions = new HashMap<>();
    private final Map<Long, HashSet<Object>> deviceReferences = new
HashMap<>();

    @Inject
    public CacheManager(Config config, Storage storage, BroadcastService
broadcastService) throws StorageException {
        this.config = config;
        this.storage = storage;
        this.broadcastService = broadcastService;
        server = storage.getObject(Server.class, new Request(new
Columns.All()));
        broadcastService.registerListener(this);
    }

    @Override
    public String toString() {
        return graph.toString();
    }

    public Config getConfig() {
        return config;
    }

    public <T extends BaseModel> T getObject(Class<T> clazz, long id) {
        try {
            lock.readLock().lock();
            return graph.getObject(clazz, id);
```

```
        } finally {
            lock.readLock().unlock();
        }
    }

    public <T extends BaseModel> Set<T> getDeviceObjects(long deviceId,
Class<T> clazz) {
        try {
            lock.readLock().lock();
            return graph.getObjects(Device.class, deviceId, clazz,
Set.of(Group.class), true)
                    .collect(Collectors.toUnmodifiableSet());
        } finally {
            lock.readLock().unlock();
        }
    }

    public Position getPosition(long deviceId) {
        try {
            lock.readLock().lock();
            return devicePositions.get(deviceId);
        } finally {
            lock.readLock().unlock();
        }
    }

    public Server getServer() {
        try {
            lock.readLock().lock();
            return server;
        } finally {
            lock.readLock().unlock();
        }
    }

    public Set<User> getNotificationUsers(long notificationId, long
deviceId) {
        try {
            lock.readLock().lock();
            Set<User> deviceUsers = getDeviceObjects(deviceId,
User.class);
            return graph.getObjects(Notification.class, notificationId,
User.class, Set.of(), false)
                    .filter(deviceUsers::contains)
                    .collect(Collectors.toUnmodifiableSet());
```

```
        } finally {
            lock.readLock().unlock();
        }
    }

    public Set<Notification> getDeviceNotifications(long deviceId) {
        try {
            lock.readLock().lock();
            var direct = graph.getObjects(Device.class, deviceId,
Notification.class, Set.of(Group.class), true)
                    .map(BaseModel::getId)
                    .collect(Collectors.toUnmodifiableSet());
            return graph.getObjects(Device.class, deviceId,
Notification.class, Set.of(Group.class, User.class), true)
                    .filter(notification -> notification.getAlways() ||
direct.contains(notification.getId()))
                    .collect(Collectors.toUnmodifiableSet());
        } finally {
            lock.readLock().unlock();
        }
    }

    public void addDevice(long deviceId, Object key) throws Exception {
        try {
            lock.writeLock().lock();
            var references = deviceReferences.computeIfAbsent(deviceId, k
-> new HashSet<>());
            if (references.isEmpty()) {
                Device device = storage.getObject(Device.class, new
Request(
                        new Columns.All(), new Condition.Equals("id",
deviceId)));
                graph.addObject(device);
                initializeCache(device);
                if (device.getPositionId() > 0) {
                    devicePositions.put(deviceId,
storage.getObject(Position.class, new Request(
                            new Columns.All(), new Condition.Equals("id",
device.getPositionId()))));
                }
            }
            references.add(key);
            LOGGER.debug("Cache add device {} references {} key {}",
deviceId, references.size(), key);
        } finally {
```

```
            lock.writeLock().unlock();
        }
    }

    public void removeDevice(long deviceId, Object key) {
        try {
            lock.writeLock().lock();
            var references = deviceReferences.computeIfAbsent(deviceId, k
-> new HashSet<>());
            references.remove(key);
            if (references.isEmpty()) {
                graph.removeObject(Device.class, deviceId);
                devicePositions.remove(deviceId);
                deviceReferences.remove(deviceId);
            }
            LOGGER.debug("Cache remove device {} references {} key {}",
deviceId, references.size(), key);
        } finally {
            lock.writeLock().unlock();
        }
    }

    public void updatePosition(Position position) {
        try {
            lock.writeLock().lock();
            if (deviceReferences.containsKey(position.getDeviceId())) {
                devicePositions.put(position.getDeviceId(), position);
            }
        } finally {
            lock.writeLock().unlock();
        }
    }

    @Override
    public <T extends BaseModel> void invalidateObject(
            boolean local, Class<T> clazz, long id, ObjectOperation
operation) throws Exception {
        if (local) {
            broadcastService.invalidateObject(true, clazz, id, operation);
        }

        if (operation == ObjectOperation.DELETE) {
            graph.removeObject(clazz, id);
        }
        if (operation != ObjectOperation.UPDATE) {
```

```
            return;
        }

        if (clazz.equals(Server.class)) {
            server = storage.getObject(Server.class, new Request(new
Columns.All()));
            return;
        }

        var after = storage.getObject(clazz, new Request(new
Columns.All(), new Condition.Equals("id", id)));
        if (after == null) {
            return;
        }
        var before = getObject(after.getClass(), after.getId());
        if (before == null) {
            return;
        }

        if (after instanceof GroupedModel) {
            long beforeGroupId = ((GroupedModel) before).getGroupId();
            long afterGroupId = ((GroupedModel) after).getGroupId();
            if (beforeGroupId != afterGroupId) {
                if (beforeGroupId > 0) {
                    invalidatePermission(clazz, id, Group.class,
beforeGroupId, false);
                }
                if (afterGroupId > 0) {
                    invalidatePermission(clazz, id, Group.class,
afterGroupId, true);
                }
            }
        } else if (after instanceof Schedulable) {
            long beforeCalendarId = ((Schedulable)
before).getCalendarId();
            long afterCalendarId = ((Schedulable) after).getCalendarId();
            if (beforeCalendarId != afterCalendarId) {
                if (beforeCalendarId > 0) {
                    invalidatePermission(clazz, id, Calendar.class,
beforeCalendarId, false);
                }
                if (afterCalendarId > 0) {
                    invalidatePermission(clazz, id, Calendar.class,
afterCalendarId, true);
                }
```

```
            }
            // TODO handle notification always change
        }

        graph.updateObject(after);
    }

    @Override
    public <T1 extends BaseModel, T2 extends BaseModel> void
invalidatePermission(
            boolean local, Class<T1> clazz1, long id1, Class<T2> clazz2,
long id2, boolean link) throws Exception {
        if (local) {
            broadcastService.invalidatePermission(true, clazz1, id1,
clazz2, id2, link);
        }

        if (clazz1.equals(User.class) &&
GroupedModel.class.isAssignableFrom(clazz2)) {
            invalidatePermission(clazz2, id2, clazz1, id1, link);
        } else {
            invalidatePermission(clazz1, id1, clazz2, id2, link);
        }
    }

    private <T1 extends BaseModel, T2 extends BaseModel> void
invalidatePermission(
            Class<T1> fromClass, long fromId, Class<T2> toClass, long
toId, boolean link) throws Exception {

        boolean groupLink = GroupedModel.class.isAssignableFrom(fromClass)
&& toClass.equals(Group.class);
        boolean calendarLink =
Schedulable.class.isAssignableFrom(fromClass) &&
toClass.equals(Calendar.class);
        boolean userLink = fromClass.equals(User.class) &&
toClass.equals(Notification.class);

        boolean groupedLinks =
GroupedModel.class.isAssignableFrom(fromClass)
                && (GROUPED_CLASSES.contains(toClass) ||
toClass.equals(User.class));

        if (!groupLink && !calendarLink && !userLink && !groupedLinks) {
            return;
```

```
        }

        if (link) {
            BaseModel object = storage.getObject(toClass, new Request(
                    new Columns.All(), new Condition.Equals("id", toId)));
            if (!graph.addLink(fromClass, fromId, object)) {
                initializeCache(object);
            }
        } else {
            graph.removeLink(fromClass, fromId, toClass, toId);
        }
    }

    private void initializeCache(BaseModel object) throws Exception {
        if (object instanceof User) {
            for (Permission permission :
storage.getPermissions(User.class, Notification.class)) {
                if (permission.getOwnerId() == object.getId()) {
                    invalidatePermission(
                            permission.getOwnerClass(),
permission.getOwnerId(),
                            permission.getPropertyClass(),
permission.getPropertyId(), true);
                }
            }
        } else {
            if (object instanceof GroupedModel groupedModel) {
                long groupId = groupedModel.getGroupId();
                if (groupId > 0) {
                    invalidatePermission(object.getClass(),
object.getId(), Group.class, groupId, true);
                }

                for (Permission permission :
storage.getPermissions(User.class, object.getClass())) {
                    if (permission.getPropertyId() == object.getId()) {
                        invalidatePermission(
                                object.getClass(), object.getId(),
User.class, permission.getOwnerId(), true);
                    }
                }

                for (Class<? extends BaseModel> clazz : GROUPED_CLASSES) {
                    for (Permission permission :
storage.getPermissions(object.getClass(), clazz)) {
```

```
                        if (permission.getOwnerId() == object.getId()) {
                            invalidatePermission(
                                    object.getClass(), object.getId(),
clazz, permission.getPropertyId(), true);
                        }
                    }
                }
            }

            if (object instanceof Schedulable schedulable) {
                long calendarId = schedulable.getCalendarId();
                if (calendarId > 0) {
                    invalidatePermission(object.getClass(),
object.getId(), Calendar.class, calendarId, true);
                }
            }
        }
    }

}
```

El CacheManager es una clase diseñada para gestionar un sistema de caché en una aplicación. Su principal función es almacenar y gestionar en memoria objetos de tipo BaseModel, así como las posiciones de los dispositivos y sus referencias. Esto permite un acceso rápido y eficiente a estos datos, reduciendo la necesidad de consultas frecuentes a la base de datos.

Funciones Clave:

1. Gestión de Caché de Objetos:
    - Mantiene un CacheGraph que contiene una representación en caché de los objetos almacenados en la base de datos. Permite la obtención y actualización de estos objetos de manera rápida mediante el uso de bloqueos de lectura/escritura para manejar la concurrencia.
2. Manejo de Dispositivos:
    - Almacena y actualiza la información sobre dispositivos, incluyendo sus posiciones actuales y referencias asociadas. Si un dispositivo se agrega o se elimina, el CacheManager actualiza la caché en consecuencia.
3. Acceso a Datos de Notificaciones:
    - Permite obtener usuarios notificados por ciertas notificaciones y las notificaciones asociadas a un dispositivo específico, facilitando la gestión de notificaciones en el sistema.
4. Sincronización de Datos:

    - Usa un mecanismo de bloqueo (ReadWriteLock) para garantizar la integridad de los datos mientras se realizan operaciones de lectura y escritura simultáneas.
5. Inyección de Dependencias:
    - Se inyectan dependencias como Config, Storage, y BroadcastService para acceder a la configuración, almacenamiento y servicios de broadcast, respectivamente.
6. Manejo de Permisos:
    - Actualiza la caché cuando se producen cambios en los permisos o en objetos agrupados, asegurando que la caché refleje las relaciones y permisos actuales.
7. Actualización y Invalidación:
    - Proporciona métodos para invalidar y actualizar objetos en la caché en respuesta a operaciones CRUD, asegurando que los datos en caché se mantengan sincronizados con la base de datos.

- **import org.traccar.sms.SmsManager;**

```
package org.traccar.sms;

import org.traccar.notification.MessageException;

public interface SmsManager {

    void sendMessage(String destAddress, String message, boolean
command) throws MessageException;

}
```

- **Interfaz de Envío de SMS:** Proporciona un contrato para las implementaciones que se encargan del envío de mensajes SMS, permitiendo que diferentes servicios o implementaciones puedan ser intercambiables sin cambiar el código que usa esta interfaz.
- **Extensibilidad:** Facilita la implementación de distintas estrategias o servicios para el envío de SMS, sin que el código cliente tenga que preocuparse por los detalles específicos de la implementación.

En resumen, la interfaz SmsManager define la estructura básica para enviar mensajes SMS y manejar posibles errores asociados con el proceso.

- **import org.traccar.storage.StorageException;**

```java
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);
    }

}
```

Manejo de Errores de Almacenamiento: La clase StorageException proporciona un mecanismo para informar sobre problemas específicos que ocurren durante las operaciones de almacenamiento, como errores al acceder o manipular datos en un almacenamiento persistente.

Flexibilidad: Al permitir mensajes personalizados y causas subyacentes, facilita la inclusión de detalles específicos sobre el error y el contexto en el que ocurrió, lo que ayuda a los desarrolladores a diagnosticar y solucionar problemas de manera más efectiva.

En resumen, StorageException es una excepción especializada que permite manejar y reportar errores relacionados con las operaciones de almacenamiento en la aplicación, ofreciendo constructores para proporcionar mensajes y causas detalladas.

- **import org.traccar.storage.query.Columns;**

```java
package org.traccar.storage.query;

import org.traccar.storage.QueryIgnore;

import java.beans.Introspector;
import java.lang.reflect.Method;
```

```java
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().sub
string(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());
        }
    }

}
```

La clase Columns es una clase abstracta que proporciona métodos para obtener una lista de nombres de columnas desde una clase dada. Estos nombres se basan en los métodos de la clase que siguen un cierto patrón (por ejemplo, métodos get o set). La clase y sus subclases permiten incluir o excluir ciertos métodos para la generación de nombres de columnas.

Columns y sus subclases proporcionan flexibilidad para obtener listas de nombres de columnas de una clase basadas en los métodos get y set. La clase abstracta define un método base que debe ser implementado por sus subclases, y las subclases permiten incluir o excluir columnas según sea necesario. Esto es útil para casos donde se necesita trabajar con datos reflejados desde las clases y manejar las columnas de manera dinámica.

- **import org.traccar.storage.query.Condition;**

```
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;
        }
    }

}
```

El paquete org.traccar.storage.query define una interfaz Condition para modelar condiciones en consultas. Aquí se destacan sus componentes principales:

- **Condition Interface**: Permite representar diferentes tipos de condiciones de consulta y combinar condiciones usando operadores lógicos.
- **Equals Class**: Define una condición de igualdad entre una columna y un valor.
- **Compare Class**: Representa una condición de comparación general con un operador específico.
- **Between Class**: Permite especificar un rango de valores para una columna.
- **Or and And Classes**: Representan condiciones combinadas usando los operadores lógicos OR y AND.
- **Permission Class**: Modela condiciones relacionadas con permisos de acceso a recursos, incluyendo la posibilidad de excluir grupos.
- **LatestPositions Class**: Define una condición para obtener las posiciones más recientes de un dispositivo específico.

El método estático merge combina una lista de condiciones en una única condición utilizando el operador AND si hay más de una.

- **import org.traccar.storage.query.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;
    }

}
```

El paquete org.traccar.storage.query define la clase Request, que encapsula los parámetros utilizados para realizar consultas en una base de datos. Los componentes clave de esta clase son:

- **columns**: Define las columnas que se deben incluir en la consulta. Utiliza la clase Columns para especificar qué columnas se deben recuperar.
- **condition**: Representa las condiciones de la consulta, utilizando la clase Condition para filtrar los resultados según ciertos criterios.
- **order**: Especifica el orden de los resultados de la consulta, utilizando la clase Order (aunque no se detalla en el código proporcionado, normalmente define cómo deben ordenarse los resultados).

Constructores: La clase tiene varios constructores que permiten crear un objeto Request con diferentes combinaciones de parámetros:

- Solo con columnas.
- Solo con condiciones.
- Con columnas y condiciones.
- Con columnas y orden.
- Con columnas, condiciones y orden.

Métodos:

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

Esta clase proporciona una forma flexible de construir solicitudes de consulta con diferentes combinaciones de parámetros.

4. Recomendaciones

Para mejorar la eficiencia en la gestión de datos y consultas, se podría implementar un sistema de monitoreo que permita observar el rendimiento en tiempo real y detectar posibles problemas. Además, se debería considerar la integración de técnicas de optimización como la indexación de columnas utilizadas frecuentemente en consultas, lo cual puede reducir el tiempo de respuesta y mejorar la eficiencia global del sistema.

5. Conclusiones

Las implementaciones de gestión de caché y consultas son cruciales para desarrollar aplicaciones eficientes y seguras. Las clases y métodos descritos proporcionan un marco robusto para el manejo de datos, el envío de mensajes y la construcción de consultas en Java. Comprender y aplicar correctamente estas técnicas permite a los desarrolladores crear sistemas más efectivos y confiables, mejorando el rendimiento y la calidad del software.

6. Referencias

Bloch, J. (2018). Effective Java (3rd ed.). Addison-Wesley.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.

Estas referencias proporcionan una base sólida para la compr