Componente Java para la integración con Callejero [WIP]

Dirección General de Sistemas de Información y Comunicaciones

Áreas de Gobierno Tecnológico de SSII y del Dato

Servicio Andaluz de Salud
Consejería de Sanidad, Presidencia y Emergencias

Contenido

Resumen

- **Versión:** v01r17
- **Fecha publicación:** 14 oct 2024
- **Entrada en vigor desde:** 15 jun 2018

blocked URL

Cumplimiento normativo

Las normas expuestas son de obligado cumplimiento. La STIC podrá estudiar los casos excepcionales los cuales serán gestionados a través de los responsables del proyecto correspondiente y autorizados por el Área de Gobernanza de la STIC. Asimismo cualquier aspecto no recogido en estas normas deberá regirse en primera instancia por las guías técnicas correspondientes al esquema nacional de seguridad y esquema nacional de interoperabilidad según correspondencia y en su defecto a los marcos normativos y de desarrollo software establecidos por la Junta de Andalucía, debiendo ser puesto de manifiesto ante la STIC.

La STIC se reserva el derecho a la modificación de la norma sin previo aviso, tras lo cual, notificará del cambio a los actores implicados para su adopción inmediata según la planificación de cada proyecto.

En el caso de que algún actor considere conveniente y/o necesario el incumplimiento de alguna de las normas y /o recomendaciones, deberá aportar previamente la correspondiente justificación fehaciente documentada de la solución alternativa propuesta, así como toda aquella documentación que le sea requerida por la STIC para proceder a su validación técnica.

Contacto Arquitectura: l-arquitectura.stic@juntadeandalucia.es

Histórico de cambios

Los cambios en la normativa vendrán acompañados de un registro de las modificaciones. De este modo se podrá realizar un seguimiento y consultar su evolución. Ordenándose de mas recientes a menos recientes, prestando especial cuidado a las cabezeras de la tablas dónde se indican las fechas de entrada en vigor y versión.

Versión	v01r1	**Fecha publicación**	15 jun 2018	**Fecha entrada en vigor**	15 jun 2018
Alcance					
• Versión inicial sobre el componente que permite la integración con Callejero fácilmente					

1. Introducción

En el presente documento se presenta la librería "**callejeroApiClient**", librería Java desarrollada por el SAS para facilitar el acceso a la información de callejero utilizada por la aplicación BDU, así como unificar en el componente el consumo de los servicios del Callejero Digital de Andalucía Unificado por terceras aplicaciones desarrolladas dentro de la organización.

Con esta librería los proveedores podrán abstraerse de los mecanismos de conexión existentes actualmente con la aplicación BDU, y CDAU, facilitando considerablemente el acceso a la información de la misma y, lo que es más importante, conseguir desacoplar los aplicativos finales, permitiendo que en un futuro se pueda evolucionar sin impactar en el conjunto de aplicaciones que hacen uso de la misma.

> ⚠ **Nota sobre el versionado dual (Ramas 1.x vs 2.x)**
>
> Para dar soporte a las diferentes plataformas tecnológicas del SAS, la librería mantiene dos líneas de desarrollo concurrentes. La rama **2.x** está diseñada para mantener la retrocompatibilidad con entornos **JDK 1.7 / Java EE**. Por su parte, la rama **2.x** está adaptada a las nuevas arquitecturas basadas en **JDK 21 / Jakarta EE**. Ambas ramas son funcionalmente equivalentes y evolucionan en paralelo en cuanto a contratos y servicios públicos se refiere, siendo la 2.0.0 equivalente a la base funcional de la 1.0.0, y nivelándose progresivamente en versiones posteriores.

2. Dependencias

CallejeroApiClient (JDK 1.7)	MacoApiClient	sas-utils
1.0.0	2.0.0	2.0.2

3. Historial de Cambios

- **Servicio añadido: ViaService**
  - FutureTask<Via> getViaByCodigo(Context c, String codigo);
  - FutureTask<List<Via>> getViaByNombre(Context c, String nombre);
  - FutureTask<List<Via>> getViaByTipo(Context c, String tipo);
  - FutureTask<Via> getViaByTipoAndMunicipio(Context c, String tipo, Long municipio);
  - FutureTask<Via> getViaByNombreAndTipoAndMunicipio(Context c, String nombre, String tipo, Long municipio);
  - FutureTask<List<Via>> getVia(Context context);

4. Servicios ofrecidos

BDU (ViaService)

Método	Descripción
FutureTask<Via> getViaByCodigo(Context c, String codigo)	
FutureTask<List<Via>> getViaByNombre(Context c, String nombre)	
FutureTask<List<Via>> getViaByTipo(Context c, String tipo)	
FutureTask<Via> getViaByTipoAndMunicipio(Context c, String tipo, Long municipio)	
FutureTask<Via> getViaByNombreAndTipoAndMunicipio(Context c, String nombre, String tipo, Long municipio)	
FutureTask<List<Via>> getVia(Context context)	

5. Arquitectura de la librería

Puesto que a fecha de realización del presente documento los servicios que se ofrecen para acceder a la información no permiten acceder a todo el catálogo de entidades almacenadas, la única forma de acceder actualmente a dicha información es haciendo uso de Réplicas de base de datos.

Ya que el acceso a la información a través de réplicas de base de datos no está aconsejado por la gran dependencia que genera entre diferentes productos, la librería de acceso a la Callejero se ha desarrollado siguiendo un modelo de API/Implementación que permite exponer a los usuarios de la librería una API única que debe ser usada y una serie de implementaciones de dicha API, las cuales serán las verdaderas encargadas de acceder a la información.

Siguiendo este patrón se ha iniciado el desarrollo con una única implementación haciendo uso de réplicas (por ser la única forma posible actualmente), esperando que en un futuro se generen nuevas implementaciones que hagan uso de distintos métodos de acceso (SOAP, REST, etc.). Los productos que estén ya desarrollados y haciendo uso de la API estándar podrán cambiar fácilmente de implementación si que el producto se vea afectado.

Requerimientos

Versión < 2.0.0

Siguiendo la normativa vigente para desarrollo de aplicaciones Java en el SAS, la librería se ha implementado con Java 7 y ha sido testada en Oracle Weblogic 12.1.3, donde se ha creado proyectos de prueba que incorporan la librería a través de Maven.

Versión >= 2.0.0

Siguiendo la normativa vigente para desarrollo de aplicaciones Java en el SAS, la librería se ha implementado con Java 21, Jakarta 10 y Microprofile 6.1 y ha sido testada en Openliberty 24.X.X.X, donde se ha creado proyectos de prueba que incorporan la librería a través de Maven.

Instalación

La librería desarrollada se encuentra desplegada en el repositorio de artefactos corporativo pudiendo ser incorporada en nuestro proyecto haciendo uso de maven:

Versión < 2.0.0

Pom Padre

```
             <dependencyManagement>
         <dependencies>
                 <dependency>
                <groupId>es.ja.csalud.sas.componentescomunes.callejeroApiClient</groupId>
                <artifactId>callejeroApiClient</artifactId>
                <version>${callejeroapi.version}</version>
                <type>pom</type>
                <scope>import</scope>
        </dependency>
          <dependencies>
 <dependencyManagement>
```

Pom módulo uso

```
            <!-- Añadir en caso de hacer uso de la interfaz-->
<dependencies>
          <dependency>
            <groupId>es.ja.csalud.sas.componentescomunes.callejeroApiClient</groupId>
            <artifactId>callejeroApiClient-api</artifactId>
            <version>${callejeroapi.version}</version>
        </dependency>
<!-- Añadir en caso de hacer uso de la implementación-->
        <dependency>
            <groupId>es.ja.csalud.sas.componentescomunes.callejeroApiClient</groupId>
            <artifactId>callejeroApiClient-jpa</artifactId>
            <version>${callejeroapi.version}</version>
        </dependency>
   <dependencies>
```

Configuración

Antes de comenzar a configurar nuestro proyecto, conviene recordar que la implementación de la API desarrollada con la que se cuenta actualmente se basa en acceso a la información a partir de un origen de datos específico para el acceso a sus datos, mientras que por otro lado la aplicación podrá tener otro origen de datos distinto para acceder a sus datos, de esta forma la aplicación puede estar en una base de datos mientras que la información sea recuperada de otra distinta.

Dicho esto, y siendo consciente de que existen diferentes formas de hacer lo mismo, a continuación se expone una serie de instrucciones a seguir para configurar un proyecto web que haga uso de esta librería y que se ejecutará en un contenedor Weblogic:

Configurar origen de datos

Versión < 2.0.0

En Weblogic: El primer paso a seguir será configurar el origen de datos en Weblogic, donde debemos crear un origen de datos que tenga visibilidad con las tablas de réplicas alojados en un esquema llamado REP_PRO_XXX. Por ejemplo, nuestro origen de datos podría llamarse "jdbc/Diraya-Callejero-DS"

Versión >= 2.0.0

En Openliberty >= 24.X.X.X:

server.xml

```
            <server>
....
<include location="persistence-config-oracle-callejero.xml" /> <!-- Se puede modificar el nombre del
fichero por el que resulte más adecuado para el ejemplo usaremos este -->
....
</server>
```

persistence-config-oracle-estructura.xml

```
            <server>
    <!-- Oracle driver -->
    <jdbcDriver id="oracle19-driver" libraryRef="oracle19-library" />

    <!-- Oracle driver library -->
    <library id="oracle19-library">
        <fileset dir="${server.config.dir}/lib/oracle" includes="ojdbc*.jar" />
    </library>

    <!-- Datasource -->
    <dataSource id="oracle" jndiName="jdbc/Diraya-Bdu-DS" queryTimeout="20s">
        <jdbcDriver libraryRef="oracle19-library"/>
        <properties.oracle URL="${jdbc.url}" user="${jdbc.user}" password="${jdbc.password}" /> <!--
queda a criterio del programador determinar las variables correctas. En caso de contenerizar el
servidor, se deberá hacer uso de las secret y configmap descritos en el helm de despliegue -->
    </dataSource>

</server>
```

Para que funcione la configuración de conexión a la bbdd se deben seguir las pautas del readme.md del arquetipo. En resumen:

pom.xml del boot

```
              <plugins>
            <plugin>
                <groupId>io.openliberty.tools</groupId>
                <artifactId>liberty-maven-plugin</artifactId>
                <configuration>
                    <copyDependencies>
                         <dependencyGroup>
                            <location>lib/oracle</location>
                            <dependency>
                                <groupId>com.oracle.database.jdbc</groupId>
                                <artifactId>ojdbc11</artifactId>
                                <version>${oracle.version}</version>
                            </dependency>
                        </dependencyGroup>
                    </copyDependencies>
                </configuration>
            </plugin>
```

Configurar persistence.xml: Configurar el fichero persistence.xml de nuestro proyecto web para añadir el orígen de datos configurado previamente. En este fichero se le asignara un nombre a la unidad de persistencia que posteriormente usará nuestro código a través de JPA. En el siguiente extracto puede verse como se crea una unidad de persistencia llamada "Diraya-Callejero-Persistence-Unit" y que es mapeada con el origen de datos que creamos previamente:

```
        <persistence-unit name="Diraya-Callejero-Persistence-Unit" transaction-type="JTA">
    <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
    <jta-data-source>jdbc/Diraya-Callejero-DS</jta-data-source>
    <mapping-file>META-INF/callejero-bd-replica.xml</mapping-file>

    <class>es.ja.csalud.sas.componentescomunes.callejero.impl.*</class>
    <exclude-unlisted-classes>true</exclude-unlisted-classes>
</persistence-unit>
```

Es importante destacar que para aislar la unidad de persistencia del resto de la aplicación y evitar posibles conflictos, es recomendable añadir la etiqueta "class" y "exclude-unlisted-classes" con el valor que puede verse en el ejemplo para que la unidad de persistencia sólo actúe sobre el paquete indicado.
En cuanto al fichero de mapeo de entidades, pueden usarse cualquiera de los siguientes mapeos:

- **META-INF/callejero-bd-replica.xml**: Cuando nuestra cadena de conexión apunte a una base de datos donde existe una réplica en el esquema REP_PRO_XXX. **Este será la configuración usada en la mayoría de los casos**
- **META-INF/callejero-bd-original.xml**: Cuando nuestra cadena de conexión apunte directamente a la base de datos
- Cualquier otra configuración requerirá que el fichero xml sea redefinido por la aplicación que hace uso de la librería

Asignar EntityManager a la librería : Una vez configurada la unidad de persistencia sólo nos quedaría indicarle a la librería cuál va a ser el contexto de persistencia que queremos que use, ya que como se ha mencionado previamente, nuestra aplicación puede tener varios orígenes de datos distintos. Para ello nuevamente existen diferentes formas de hacerlo, a continuación se muestra una manera muy simple consistente en crear una clase que es ejecutada al iniciar nuestro servidor y se encarga de asignarle a la librería el contexto de persistencia a usar. A partir de este momento nuestra librería es completamente funcional.

Versión 1.0.0

Callejero EntityManager Config

```
            @Singleton
@Startup
public class DaoEntityManagerConfig {

    @PersistenceContext(unitName = "Diraya-Callejero-Persistence-Unit")
    private EntityManager callejeroEntityManager;

    @PostConstruct
    public void init() {
        EntityManagerContainer.setEntityManager(callejeroEntityManager);
    }

}
```

Versión >= 2.0.0

Callejero EntityManager Config

```
            import es.ja.csalud.sas.componentescomunes.callejero.impl.jpa.control.config.
EntityManagerContainer;
import jakarta.enterprise.context.ApplicationScoped;
import jakarta.enterprise.context.Initialized;
import jakarta.enterprise.event.Observes;
import jakarta.persistence.EntityManager;
import jakarta.persistence.PersistenceContext;

@ApplicationScoped
public class EntityManagerProducer {

    @PersistenceContext(unitName = "Diraya-Callejero-Persistence-Unit")
    private EntityManager callejeroEntityManager;

    public void init(@Observes @Initialized(ApplicationScoped.class) Object pointless)  {
        EntityManagerContainer.setEntityManager(callejeroEntityManager);
    }

}
```

Generar configuración de la API : El componente funciona en base a Futures, se hace necesario el uso de un Executor encargado de gestionar los hilos que se vayan lanzando en el sistema. Para ello bastará con añadir un Produces en nuestro proyecto como el siguiente:

Versión 1.0.0

Productor Estructura Api Config

```
            @Produces
@Singleton
public CallejeroApiConfig producerCallejeroApiConfig() {
        return new CallejeroApiConfigBuilder().executorPoolSize(20).build();
}
```

Versión >= 2.0.0

Productor Estructura Api Config

```
            import es.ja.csalud.sas.componentescomunes.callejero.impl.jpa.control.config.
CallejeroApiConfig;
import jakarta.enterprise.context.ApplicationScoped;
import jakarta.enterprise.inject.Produces;
import jakarta.inject.Singleton;
import org.eclipse.microprofile.context.ManagedExecutor;
import org.eclipse.microprofile.context.ThreadContext;

@ApplicationScoped
public class CallejeroContext {
    @Produces
    @Singleton
    public CallejeroApiConfig producerCallejeroApiConfig() {
        ManagedExecutor executor = ManagedExecutor.builder()
                .propagated(ThreadContext.CDI, ThreadContext.APPLICATION)
                .maxAsync(20)
                .maxQueued(20)
                .build();
        return new CallejeroApiConfig.CallejeroApiConfigBuilder().executor(executor).executorPoolSize
(20).build();
    }
}
```

IMPORTANTE

Es especialmente importante ser consciente de la forma de funcionar de un Produces, ya que si en cada inyección se realiza una llamada al Produces se creará en cada llamada un Executor diferente, generando un problema de rendimiento en el servidor por acumulación de hilos. Por tanto, queda a responsabilidad del programador que hace uso de la librería el asegurar que sólo se genere un Executor para toda la librería o por servicio. En el presente ejemplo esto se ha conseguido haciendo uso de @Singleton, pero igualmente podría conseguirse mediante variables estáticas, etc.

6. Instrucciones de uso

Una vez configurado nuestro proyecto, el uso de la librería es tan simple como inyectar el servicio que deseemos utilizar (ViaService, etc) en cualquier parte de nuestro código para poder hacer uso de toda la funcionalidad que ofrece la librería.

La librería ofrece servicios que retornan Futures en lugar de devolver directamente las entidades solicitadas. De esta forma el programador podrá paralelizar otras tareas mientras la librería obtiene el dato, cosa que hará que mejore el rendimiento considerablemente en el caso de que el dato tenga que ser obtenido por medio de servicios webs.

7. JavaDoc y Código Fuente

Toda la documentación del proyecto en forma de JavaDoc, así como el código fuente del mismo, se encuentra desplegado al Artifactory del SAS, por lo que el programador podrá descargarlo y consultarlo para ver el detalle de todas las clases de la librería.
Se recomienda especialmente consultar JavaDoc de los servicios principales de la API (**ViaService, etc**), donde podrá ver el detalle de todas las operaciones permitidas actualmente.