# Proyecto Reposición automática

## Objetivo

El objetivo de este proceso es incorporarle al Drako de Telko una serie módulos que permita recepcionar en forma automática, los pedidos de reposición de productos.

## Definiciones del proceso de reposición

La estrategia comercial de Telko implica dejarle ciertas ópticas, una cantidad de productos en consignación, quedando a la espera de su venta por parte de las ópticas. Una vez realizada la venta, éstas deben rendir cuentas a Telko del detalle ventas realizadas, y éste, procede a reponer la mercadería y facturar (a la óptica) las ventas realizadas. Hoy en día este proceso es manual, lo cual genera demoras en la rendición de cuenta de las ópticas. Implicando la no facturación por parte de Telko de ventas que no fueron rendidas, generando atrasos financieros.

## Implementación

La idea es crear una serie de módulos los cuales algunos van a ejecutarse en los servidores de Telko (módulos backend) y otros van a estar instalados en cada óptica adherida al módulo de reposición automática (módulos frontend).

En un principio vamos a separar todo el código que generemos para estos módulos del propio código de Drako. Tal vez en un futuro podamos acoplar parte de la funcionalidad dentro del ejecutable de Drako Vision.

Vamos a separar en dos etapas el desarrollo. La primera etapa que es la que se incluye en este documento, consiste en preparar toda la infraestructura de base para que luego en la etapa 2 pasemos a implementar específicamente los requerimientos de la reposición automática.

Para que la etapa 2 se pueda realizar en forma correcta, debemos generar todo un sistema de sincronización automática de datos, para que luego toda la lógica específica de las reposiciones automáticas funcione correctamente.

## Etapa 1

En esta etapa vamos a generar el código necesario para poder realizar toda la sincronización de los datos base, es decir tablas de códigos relacionadas a los artículos.

## Base de sincronización

Se creó una nueva base de datos para gestionar la sincronización de datos, a fin de que haga de intermediaria entre la base de Telko y la de las ópticas.

Debemos utilizar dicha base de datos como almacenamiento temporal de la información que vamos a distribuir a las ópticas.

### La lógica general sería la siguiente:

Va a existir una tabla de nombre *Tenants* donde vamos a cargar todas las ópticas que estén adheridas al sistema de reposición automática.

Se debe desarrollar un módulo que revise cada cierto tiempo (parametrizable) ciertas tablas predefinidas de la base de Telko, y genere los registros que se deben enviar a cada Tenant. Debe tener la inteligencia suficiente para que solo genere aquellos registros que sufrieron cambios desde la última revisión. Dichos registros se serializarán (se propone el formato JSON) y se almacenan temporalmente en la base de sincronización, a la espera de ser enviados al tenant.

Cada tenant (óptica) tendrá un servicio corriendo en su servidor el cual estará consultando un Web Services (a desarrollar) de Telko, el cual le envíe los datos a actualizar. También en la etapa 2 el tenant enviará información de ventas a ser repuestas.

### Proceso de distribución de información.

Deberá existir un proceso que corra (en el servidor de Telko) cada x tiempo el cual debe ser parametrizable y realice las siguientes operaciones:

* Recorrer la tabla Tentant, y por cada registro activo realice las operaciones a continuación.
* Buscar la última fecha de actualización para el Tenant que estamos procesando. Dicha fecha se debe guardar en el campo de la tabla Tenant. Si es la primera vez, ésta va a estar vacía y, podemos utilizar una fecha ficta lo suficientemente antigua para que procese todos los registros (ej. 01/01/2000).
* Por cada tabla a distribuir (base de Drako Vision) , debemos obtener los registros cuya fecha de actualización sea mayor o igual a la fecha obtenida en el punto anterior. Para eso utilizaremos el campo FechaModificado de las tablas que estamos consultando. Todas las entities de Drako Vision tienen un campo llamado FechaModificado que está mapeado al campo de la base donde se almacena la fecha que se modifica el registro.
* Para saber que tablas de Drako Vision debemos distribuir, tendremos que recorrer los registros de la tabla EntitiyDistribucion. Por el momento no habrá más condiciones que la fecha de modificación pero para otras etapas podrán haber filtros que habrá que respetar ya que no necesariamente todos los registros se distribuyen a todos los tenants.
* Hay que tener en cuenta que hay tablas que tienen información agregada que se debe distribuir en conjunto., Ej. Si se distribuye un registro de la tabla Venta (DocumentoSalida), también hay que distribuir los registros de su detalle (DocSalidaDetalle). Para eso hay que revisar la tabla EntityDistribucionAgregada, donde dado una tabla madre, se indica que tablas hijas van agregadas y enviar todos los registros en una sola transacción. Esto hay que tenerlo en cuenta porque las clases DTOs deben contemplar estas agregaciones.
* Por cada tabla recorrida vamos a obtener una colección de Entities con los registros a actualizar, debemos transformar estas entities en objetos DTO y luego serializar cada objeto y guardarlo en la tabla DistribucionRegistro (campo Datos).
* Allí terminaría el proceso del servicio de generación de datos, pero ahora debemos dejar disponible un par de Web Services para consumo del tenant.
* Un método del WS debería permitirle al tenant obtener los registros a actualizar. Para esto, el tenant llama a este método (debemos definir de qué forma el tenant se identifica ante el web service) y el procedimiento del WS debería devolverle una serie de registros a actualizar correspondiente a dicho tenant.
* Antes de enviar la colección de registros, debemos armar un paquete. Para esto vamos a cargar en el campo PaqueteGuid, un Guid generado automáticamente (el mismo para todo el paquete). Guardamos en los registros y enviamos en forma serializada (hay que definir de qué forma se serializa el paquete de registros, posiblemente un string JSON) los registros del paquete.
* Luego el proceso del tenant que invoca el web service, debe deserializar el string en clases DTO, convertirlo a entities, guardarlos en la base del tenant.
* Por último, ni bien el proceso del tenant termina de procesar el paquete de datos, debe invocar a otro web services de Telko indicando que ya proceso el paquete (envía el valor GUID), y este método debe eliminar el paquete de los pendientes de distribución.

Problemática relacionada a la actualización de datos.

Existe una dificultad para lograr sincronizar los datos de las tablas de Telko y los tenants, y es que en las ópticas puede crear otros registros aparte de los que se importan de Telko. Dado que normalmente las tablas de Drako Vision tienen su clave primara autonumérica, no podemos machear los registros por estos campos. Para esto cada tabla que necesite se sincronizada deberá tener un campo de nombre IdExterno, a los cuales se le cargará el campo Id importado de Telko.

También cada vez que el proceso de sincronización del tenant proceda a guardar un registro importado de Telko, se deberá confirmar si ese registro ya existe (se genera un update) o es nuevo (se genera un insert). Para eso siempre se deberá buscar por IdExterno y nunca por Id.

# Resumen

A modo de resumen el proceso de sincronización deberá realizar lo siguiente.

1. Revisar en las tablas de Drako Vision de Telko que registros cambiaron desde la última sincronización. Este proceso se debe realizar por cada tenant.
2. Almacenar en la base de sincronización los registros (ya serializados) que se deben enviar al tenant.
3. Publicar un web service que sea invocado por el tenant y envie un paquete de registros a actualizar.
4. Publicar otro web service que sea invocado por el tenant que permita eliminar un paquete de registros enviados y procesados por el tenant.

## Base de sincronización

La base de sincronización será independiente de las bases Vision de Telko y de los Tenants. A continuación, se detalla las tablas que contendrá su primera versión.

\*EntityDistribucion

Se definirán todas las tablas que se van a distribuir a los tenants.

Campos:

* Id (clave primaria)
* Distribuir. Indica si la tabla se distribuye o es una tabla agregada (cuya distribución va a partir de su tabla madre).
* EntityName. Indica el nombre de la entity (o tabla) que se va a distribuir.
* Filtro, puede ser que determinadas tablas no se deban distribuir en su totalidad, en dicho caso se establecerá un filtro (por el momento no aplica).
* Orden. Este es un campo muy importante. El proceso deberá revisar las tablas a distribuir en el orden que aparecen.

\*EntityDistribucionAgregada

Aquí se definen las tablas agregadas.

Campos:

* Id (clave primaria)
* EntityId. El Id correspondiente a la entity madre de la tabla EntityDitribucion.
* EntityAgregadaId. El Id correspondiente a la entity hija de la tabla EntityDitribucion.

\*Tentant

Aquí se cargar las ópticas que estarán adheridas al sistema de reposición automática.

Campos:

* Id (clave primaria).
* Nombre.
* Rut, el nro. registro de la empresa.
* FechaActualizacion, este campo es fundamental. Aquí se almacenará por cada tentant, cual fue la ultima vez que se generó una distribución de datos.
* IdProveedorDrako, El Id de proveedor de la base de Tenant que se corresponde con la empresa Telko. En un primero momento no lo vamos a utilizar.

\*SubTenant

Por temas logísticos a veces Telko maneja en forma separa clientes que son en realidad la misma empresa. Pero nosotros en el sistema de sincronizacipon debemos interactuar contra sola una base de datos. Vamos a definir subtentants por cada cliente creado en Telko perteneciente a la misma empresa (mismo RUT).

Campos:

* Id (clave primaria).
* Descripción.
* TenantId, el id del tenant al que pertenece.
* ClienteDrakoId, el id de la tabla cliente de la base Vision de Telko.
* DepositoDestinoId, cuando comencemos la segunda etapa donde vamos a sincronizar ventas, debemos saber a que deposito se va a enviar la mercaderia. Aquí vamos a cargar el Id de deposito de la base del Tenant.

\*DistribucionRegistro

Aquí se van almacenando todos los registros que se detectaron que hay que distribuir a los respectivos tenants.

Campos:

* Id (clave primaria).
* TenantId, a que tenant corresponde distribuir el registro.
* EntityId, la id de la entity que el campo datos hace referencia.
* Datos. Almacena el contenido serializado del registro que se va a distribuir. Esta serialización se debe hacer a partir de una clase DTO y no desde una entity.
* FechaYHora, la fecha y hora de creación del registro (tipo timestamp).
* PaqueteGuid, un valor GUID correspondiente a un lote de registros a enviar.

## Estructura de proyectos inicial

Se proveerá una solución con una serie de proyectos iniciales (vacíos) para comenzar a trabajar en la implementación.

Son plantillas vb.net con framework 4.

Studio.Vision.Telko.Backend.Domain

Aquí vamos a crear el código correspondiente a la lógica del negocio del backend que va a correr en Telko. Incluir aquí todas las rutinas del sistema de reposiciones, correspondiente a la parte de Telko. Es del tipo librería de clases.

Studio.Vision.Telko.Backend.Service

Es un proyecto de tipo servicio de Windows que actuará como chron y se encargará de disparar los procesos de verificación de datos a distribuir. Aquí poner la menor cantidad de código posible, solo el necesario para disparar las rutinas de verificación que deberán estar implementadas en el proyecto anterior.

Studio.Vision.Telko.Backend.Test

Proyecto para testeo de los proyectos de backend, es de tipo consola.

Studio.Vision.Telko.Backend.Web

Este proyecto va a publicar los web services (del tipo WebApi restfull) que los tentants van a consumir. También (en otras etapas del desarrollo) será el encargado de publicar una aplicación web para poder cargar y consultar información de la base de sincronización. Es del tipo aplicación asp.net.

Studio.Vision.Telko.FrontEnd.Domain

Aquí vamos a crear el código correspondiente a la lógica del negocio del frontend que va a correr en cada tenant. Incluir aquí todas las rutinas correspondientes a la parte de del tenant del sistema de reposiciones. Es del tipo librería de clases.

Studio.Vision.Telko.Frontend.Service

Es un proyecto de tipo servicio de Windows que actuará como chron y se encargará de disparar los procesos de verificación del tentant y en otras etapas envío de información hacia Telko. Aquí poner la menor cantidad de código posible, solo el necesario para disparar las rutinas de verificación que deberán estar implementadas en el proyecto anterior

Studio.Vision.Telko.FrontEnd.Test

Proyecto para testeo de lor proyectos de frontend, es de tipo consola.

Studio.Vision.Telko.Shared

Proyecto de tipo librería de clases que contendrá el código común entre el backend y frontend. Ejemplo, interfases y clases DTOs.

Studio\_Telko\_Sync

Studio\_Telko\_SyncDBSpecific

Son los proyectos generados del LLBLGenPro correspondiente a la base de sincronización.

Carpeta Referencias

Contiene librerías ya desarrolladas de uso común.

Al momento se incluyen las librerías Studio.Phone.DAL.dll y Studio.Phone.DALDBSpecific.dll, que son las librerías generadas del LLBLGenPro para el manejo de la base Drako Vision. Se deben utilizar para el acceso a la base de datos tanto desde el BackEnd y FrontEnd.

### ¿Como comenzar?

Instalar dos bases de datos Vision, una correspondiente a Telko y la otra a un tenant cualquiera. Se adjuntan las bases en la carpeta metadata.

Crear las clases DTOs para poder intercambiar información. Hacerlo solo para las tablas a distribuir.

El conjunto básico de tablas a distribuir se podrá ver en la base de sincronización que se enviará. Seguramente se agregan otras tablas en el futuro.

Las clases DTO deben tener los mismos campos que las entities. Se adjuntará también el proyecto LLblGenPro para poder todos los metadatos de las entities.