Trabajo Practico Integrador de Diseño de Sistemas

Curso: K3051

Grupo: Nº1

Integrantes:

-Maximiliano Quiroga	**147.884-9**
-Bruno Bisio	**146.609-4**
-Matias Lopez Pose	**134.909-0**
-Matias Schaab	**150.368-6**
-Lucas Agustin Gonzalez	**143.156-0**

Fecha de entrega: 15/02/2017

Instalación del TP en LINUX

1-Instalar Java 8:

-Enlace Profesor:
http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
o bien instalar desde el ppa de oracle

2-Bajar e instalar Eclipse Mars

http://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/mars2

3-Instalar GitHub:

- Abrimos una terminal
- Escribimos: “sudo apt-get install git” y contraseña
- Apretamos enter y ponemos que si a todo (y)
- Clonar desde desde git en una terminal
    - Escribimos: “cd -”
    - Escribimos:”sudo git clone https://github.com/matiaslp/tp-anual-grupo1.git” y contraseña.

4-Instalar Tomcat 8.0.41

- Enlace de Tomcat: http://tomcat.apache.org/download-80.cgi#8.0.41
- Enlace de Profesor:
http://www.campusvirtual.frba.utn.edu.ar/especialidad/pluginfile.php/111447/mod_folder/content/0/DespliegueDeWebappsEnTomcat.pdf?forcedownload=1

6-Instalar MongoDB

- Enlace oficial:
https://docs.mongodb.com/manual/tutorial/install-mongodb-on-ubuntu/

7- MySQL

- Descargar y ejecutar e instalar (.deb en linux))
- Download MySQL Community Server :https://dev.mysql.com/downloads/mysql/
- MySQL Workbench: https://www.mysql.com/products/workbench/

Decisiones de Diseño

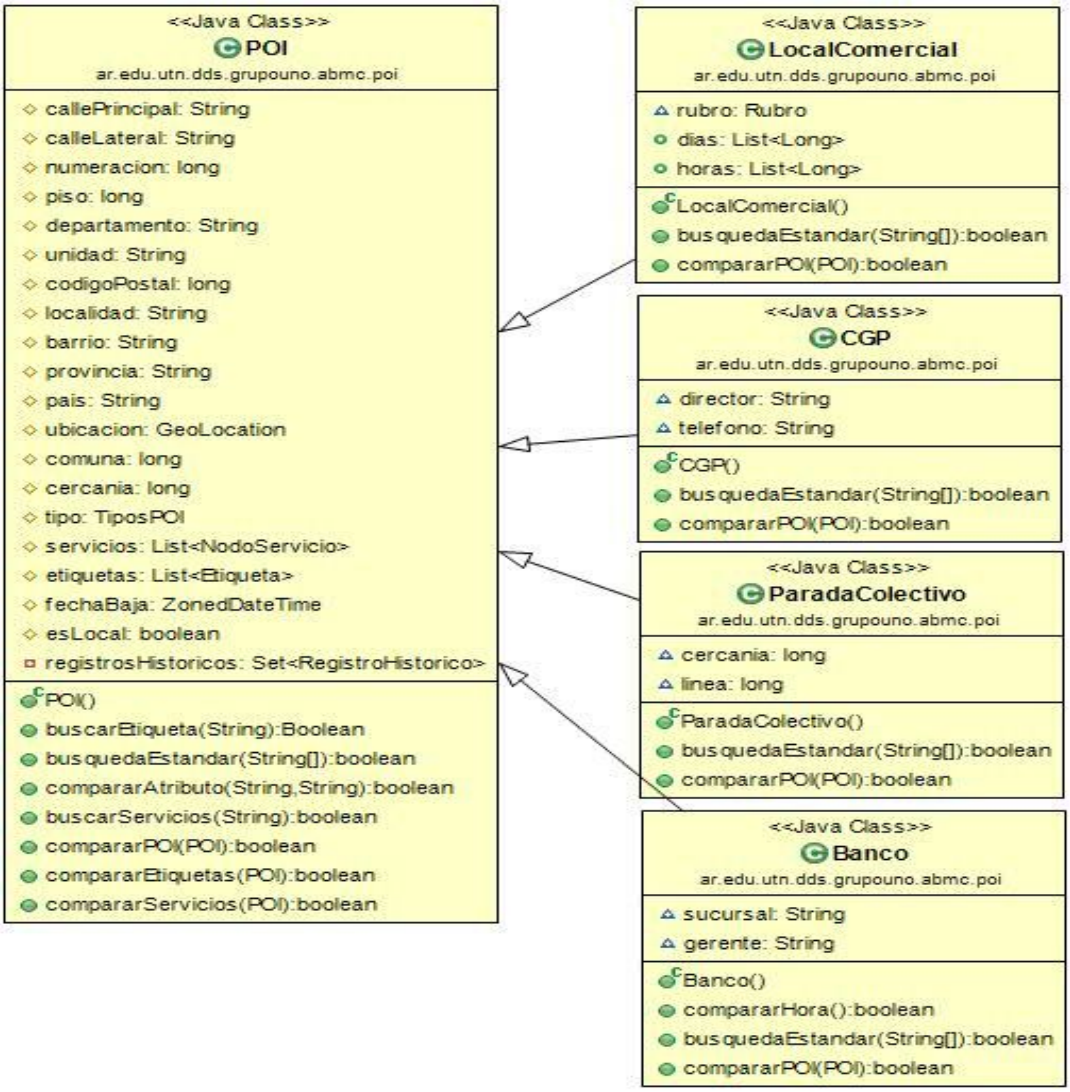
Entrega 1:

1. Decidimos utilizar el algoritmo de distancia de Leveinshtein.
2. Se evitó buscar por los valores de cercanía, ubicación (coordenadas) y días y horarios de locales comerciales.
3. Los valores numeración, piso código postal, linea y comuna los consideramos numéricos y los buscamos como "literal" (sin Leveinshtein), para evitar casos como ingresar 1er piso y que figuren en los resultados todas las paradas de líneas de colectivo con valores de 1 a 199, todas las direcciones en dicho rango, todos los pois con el campo comuna ingresado (todas las comunas que existen son en dicho rango), etc.
   Por un tema de usabilidad y escalabilidad del sistema decidimos no usar Leveinshtein en dichos campos, teniendo en cuenta la poca variación de posibilidades con numericos y la mayor probabilidad de que los mismos sea valores bajos (menores a 100)

Entrega 2:

En la entrega 1 se había diseñado la búsqueda cumpliendo únicamente los puntos detallados en los requerimientos de dicha entrega.

Para la entrega 2 ya se modifica la misma, se reestructura para que busque por todos los atributos de un poi sin importar su tipo u origen (pueden ser del servicio externo). esto se logra mediante el patrón de diseño template Method.



Al mismo tiempo, La interfaz con los servicios externos se diseña con un patrón DTO.

Se decide aprovechar también la clase POI_DTO como constructor de POIs locales por la similitud del código y para reducir redundancia en el proyecto.

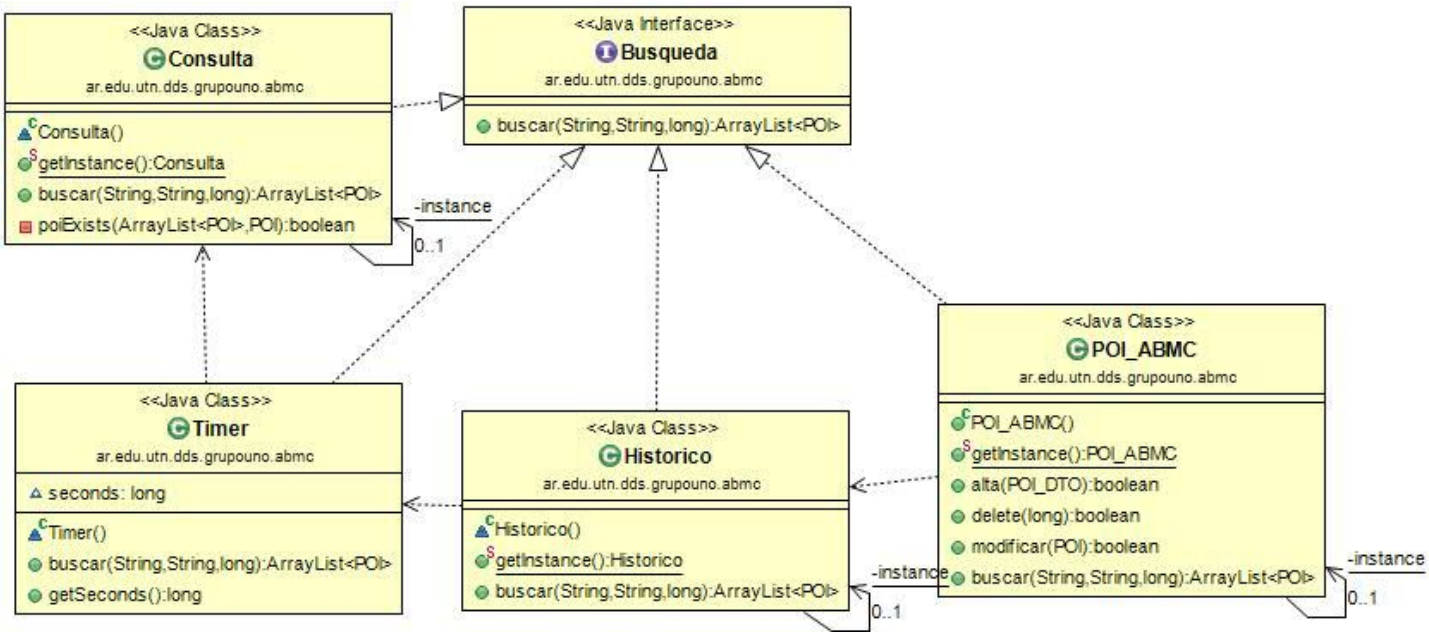
Se comienza a separar “sub-sistemas” y abstraer la implementación de servicios del resto del proyecto, el primer paso en dicho sentido se realiza al crear la clase POI_ABMC como Singleton, donde se realizan los pedidos de altas, bajas, modificaciones y búsqueda de POI.

Entrega 3:

Históricos y modificaciones a la búsqueda de POIs:

Se crea la clase DB_Historico (Singleton) para administrar los datos históricos requeridos en la entrega y como punto de acceso a los métodos de reportes.

Para abordar la necesidad de obtener tiempos de consulta así como almacenar datos históricos, se utilizó la propuesta de la guía de implementación presentada en clase. La misma se detalla en el siguiente diagrama de clases:



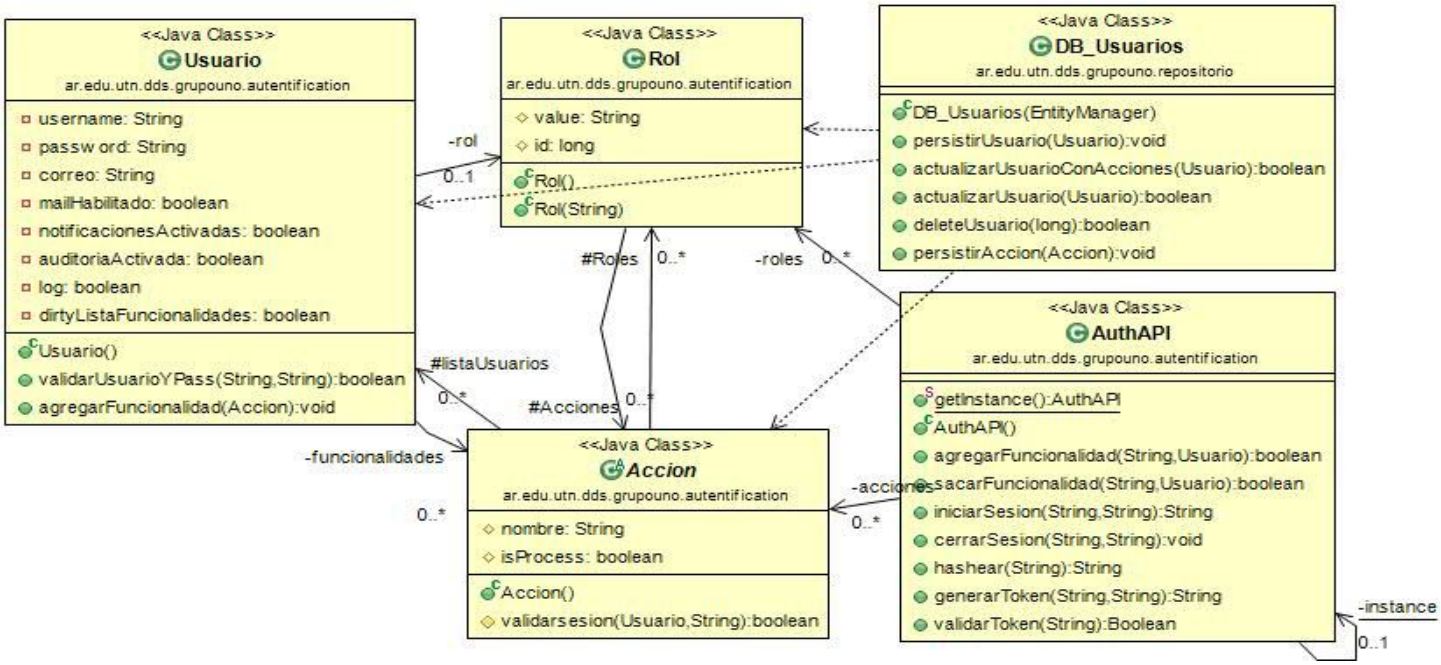
Servicio de accesos:

En lo referente a accesos, se decidió tener acciones que son permitidas a determinados roles y a su vez son asignadas a los usuarios.

- A un usuario solo pueden asignársele acciones permitidas para su rol.
- Un usuario solo puede tener un rol,
- Una acción puede estar permitida en más de un rol.

Se dispone de la clase AuthAPI (Singleton), encargada de administrar acciones, roles, tokens, accesos (login, logout).

La clase DB_Usuarios (Singleton) permite realizar altas, bajas y modificaciones de usuarios.



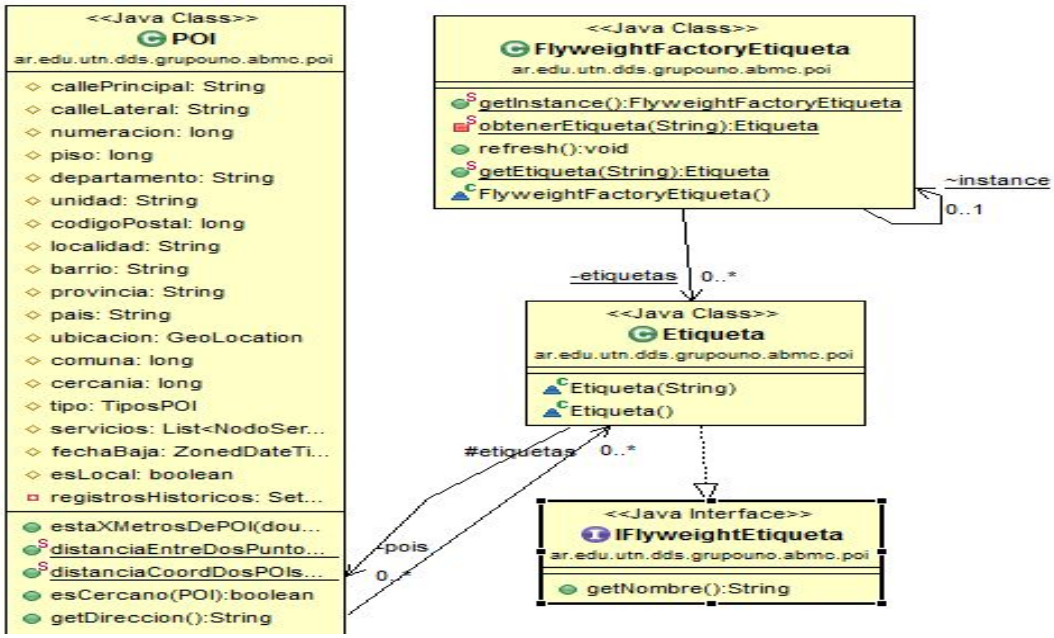
Listado de acciones

- Proceso Actualizar Local Comercial (ingresa una lista de palabras que van a ser asociadas al local comercial y desestima la lista anterior) (admin)
- Proceso Agregar Acciones (Agrega funcionalidades a los usuarios o a sí mismo -siempre que tenga permiso-) (admin)
- Proceso Baja de POIs (Ejecuta el proceso de baja de pois) (admin)
- Proceso Múltiple (permite ejecutar varios procesos) (admin)
- Búsqueda de POIs (se ingresa 1 o varias palabras y se busca por cada una con un OR inclusivo) (admin y terminal)
- Cambiar Estado Notificación (Habilita o deshabilita el envío de correo desde terminales a los administradores si tarda demasiado la búsqueda) (admin y terminal)
- Cambiar Estado Mail (Habilita o deshabilita la recepción de correos del administrador) (admin)
- Obtener Info Poi (obtiene los detalles de un poi) (admin y terminal)
- Reportes por búsqueda (admin)
- Reportes por usuario (admin)
- Reportes por fecha (admin)
- Generar Log (habilita o deshabilita los logs de búsquedas para el usuario) (admin y terminal)

Cambios en el diseño:

En POIs, en las etiquetas: Utilizamos el patrón de diseño Flyweight junto con Factory. El Flyweight lo utilizamos para no tener que repetir instancias de una misma etiqueta para diferentes POIs y así obtener un sistema más óptimo. El patrón Factory se encargaría de

gestionar los Etiquetas existentes.



Entrega 4:

Procesos:

La misma se realizó inicialmente utilizando el patrón Command, pero luego se rediseñó utilizando Quartz Scheduler.

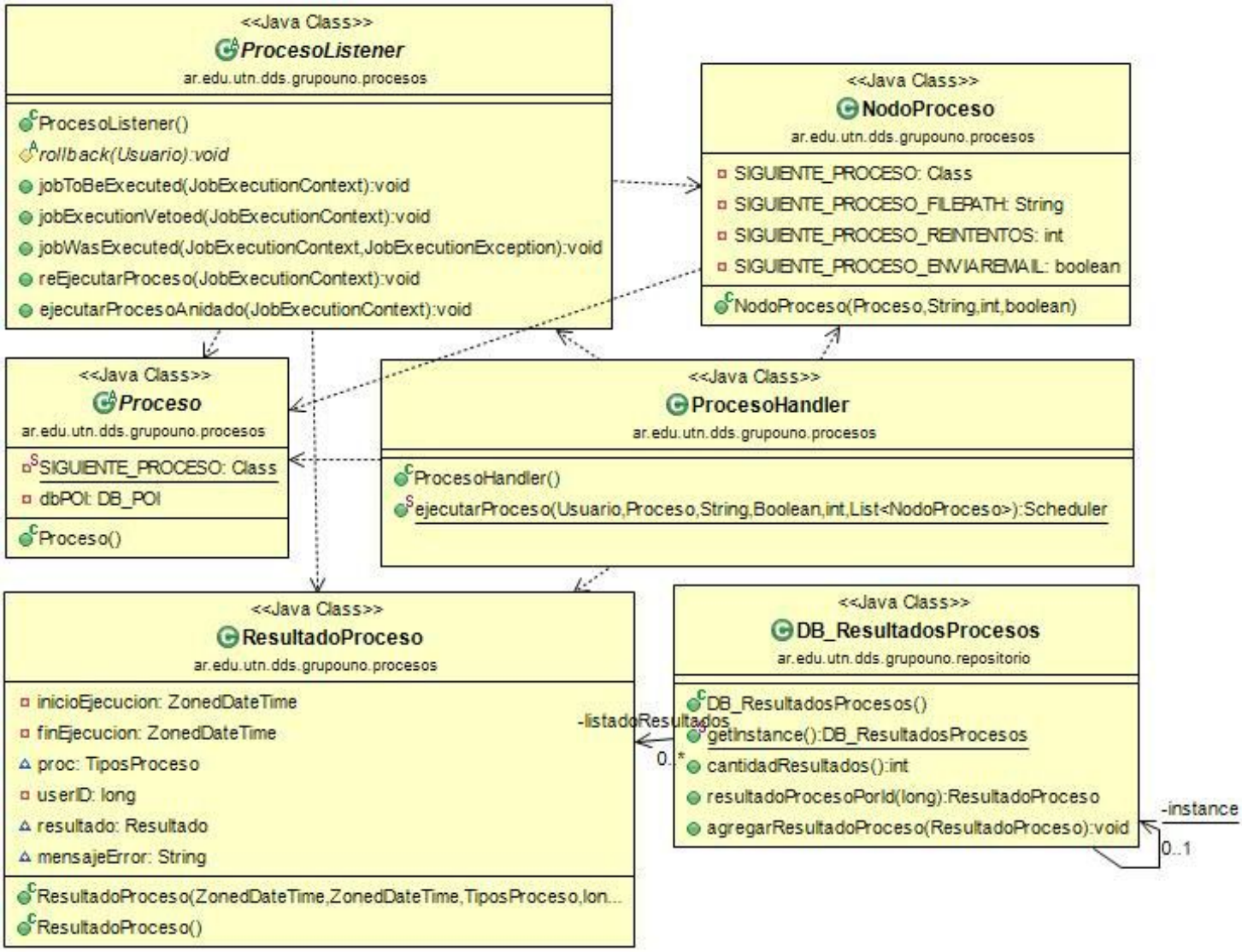
Se creó un ProcesoHandler, para la ejecución de procesos.

NodoProceso es utilizado para encolar procesos y ejecutarlos (Proceso Múltiple).

ProcesoListener se encarga de evaluar el resultado de un proceso y en caso de que sea necesario:

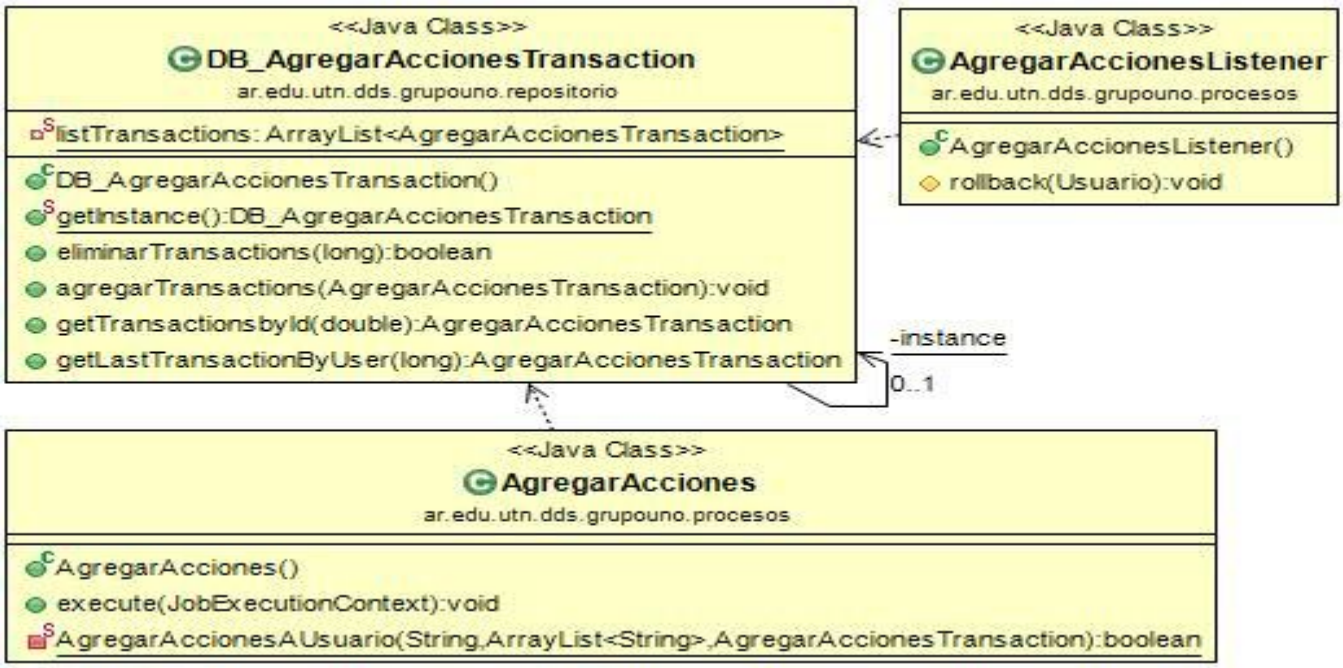
- Llama a DB_ResultadoProceso (Singleton) para almacenar resultados.

- Reejecuta un proceso si tiene reintentos pendientes.
- Llama a EnviarEmail para enviar reportes de dichos resultados.



Transacciones:

Como parte de los los requerimientos indican la necesidad de hacer el proceso AgregarAcciones reversible, se implementó una clase DB_AgregarAccionesTransaccion (Singleton), encargada de administrar la información transaccional del proceso.



Entrega 5:

Cambios en en diseño:

Se evaluó la posibilidad de utilizar AngularJS o JSF con PrimeFaces. Ante el desconocimiento de ambas tecnologías, se decidió utilizar la que más rápido nos fue posible interiorizarnos por cuestión de tiempos de entrega.

Al momento de la entrega:

Se administra correctamente el login, autentificación, y se mantiene una sesión a lo largo de las pantallas.

La mayoría de las pantallas funcionan correctamente.

Cambios al diseño por nuevos requerimientos:

Se agrega un atributo “linea” a ParadaColectivo

Se crea la clase DB_Sesion (Singleton)

Temas pendientes por falta de tiempo que hubiéramos deseado poder abordar:

Seguridad (principalmente limitar el acceso a otras pantallas por medio de URL, SSL, encriptación de passwords, entre otros)

Limitar el acceso multisesion (en la versión actual un usuario puede loguearse múltiples veces sin salir de su sesión, tampoco hay timeout de sesión)

Pantallas:

A continuación se pueden vislumbrar las pantallas realizadas en JSF con PrimeFaces:

1. LOGIN
    - 1.1. Buscar POIs
    - 1.2. Procesos
        - 1.2.1. Actualización de Locales Comerciales
        - 1.2.2. Baja de POIs
        - 1.2.3. Agregar Acciones Para Todos Los Usuarios
        - 1.2.4. Proceso Múltiple
            - 1.2.4.1. Procesos
                - 1.2.4.1.1. Actualización de Locales Comerciales
                - 1.2.4.1.2. Baja de POIs
                - 1.2.4.1.3. Agregar Acciones Para Todos Los Usuarios
    - 1.3. Reportes
        - 1.3.1. Reporte Cantidad De Resultados Por Terminal
        - 1.3.2. Reporte Busqueda Por Fecha
        - 1.3.3. Reporte Cantidad De Resultados Por Usuario
    - 1.4. Búsqueda de historial
    - 1.5. ABM Usuarios
        - 1.5.1. ALTA Usuario
        - 1.5.2. BAJA Usuario
        - 1.5.3. MODIFICACIÓN
            - 1.5.3.1. Modificación de Usuario
    - 1.6. ABM POIS
        - 1.6.1. ALTA POI
        - 1.6.2. Búsqueda de pois a dar de baja
            - 1.6.2.1. Baja POI
        - 1.6.3. Búsqueda de pois a modificar
            - 1.6.3.1. Modificación de POI
    - 1.7. Configuración de Acciones ante una Consulta
    - 1.8. Configuración

Entrega 6:

Nos decidimos por Mysql y el ODM Hibernate. Al ya tener los accesos a datos en clases con prefijo “DB_”, fue bastante sencilla la conversión.

Se creó un nuevo Singleton DB_Etiquetas encargado de administrar el acceso a base de datos por las mismas, ya que el patrón Flyweight obliga a que las mismas no sean dependientes de los POIs y no se eliminen.

Se decidió dejar DB_AgregarAccionesTransacciones fuera de la migración a Hibernate por cuestión de tiempos y por comprender que el rollback solo tendría sentido y utilidad en momentos inmediatos posteriores a la ejecución del proceso.

Se evaluaron los 3 tipos de herencia posibles con Hibernate para la herencia de POI con Banco, ParadaColectivos, LocalComercial y CGP. Se seleccionó la opción que permitía una tabla por subclase (`InheritanceType.JOINED`) ya que aporta mayor flexibilidad a la vez que utilizaba la menor cantidad de espacio en disco rígido.

Se debió modificar sustancialmente la implementación de Acciones, Roles y su relación con Usuarios para permitir que puedan ser persistidos por Hibernate.

Se vislumbra que se había diseñado teniendo en cuenta usuarios con username único. Se implementa un constructor de usuarios (UsuariosFactory) para administrar la limitación de username así como facilitar la creación de nuevos usuarios, permitiendo la asignación de determinadas acciones por defecto a nuevos usuarios a partir de su rol seleccionado.

Fue necesario agregar listas en el otro extremo de algunas relaciones de muchos a muchos. Ej: se agregó una lista de históricos en POI, ya que el historial debía contener una lista de pois encontrados. También una lista de Roles que permiten una acción en la clase de cada acción.

* Diagrama de Entidad Relación hoja Nº 24

Entrega 7:

Decidimos utilizar MongoDB junto con Morphia para esta entrega.

El punto 1 que indica que la búsqueda en el historial de consultas demora bastante, fue resuelta migrando los registros históricos a mongoDB, solucionando así la necesidad de realizar JOINs con la tabla de POIs.

Los puntos 2 y 3 requieren resolver los problemas de tiempo de demora y costos de los servicios externos. Decidimos resolverlos con una cache en mongoDB para los servicios externos.

Resumen de Patrones de Diseño:

- En Usuario: Utilizamos el patrón Factory al crear un nuevo usuario, para asegurarnos de haya una un solo usuario con ese nombre y para determinar configuraciones y funciones dependiendo del rol del usuario.
- En POI_ABMC: Utilizamos el patrón de diseño Facade ocultando algunas clases del sub-sistema de busquedaso.
- Usamos el patrón DTO para las consultas externas.
- Para la creación de POIs utilizamos el patrón de diseño Builder.
- Utilizamos Template Method en POI con locales comerciales, bancos y CGP para poder crear diferentes tipos de POIs.
- Singleton fue utilizado en varias clases para mantener una sola instancia en tiempo de ejecución, ejemplo en las clases que proveen servicios como POI_ABMC, todos los que empiezan con DB, etc.
- El patrón de diseño Model View Controller lo utilizamos en el frontend.

Herramientas utilizadas:

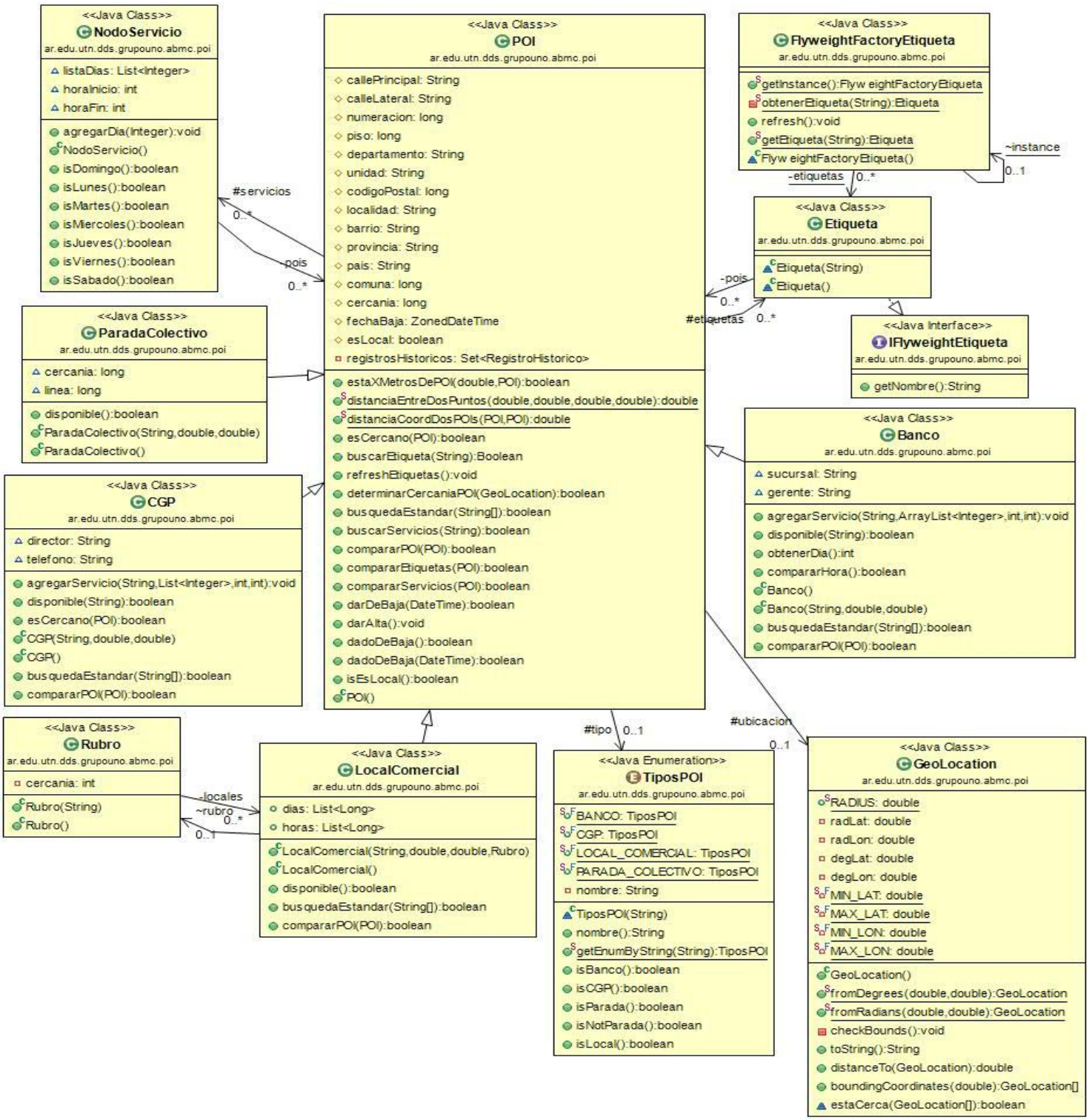
Utilizamos las siguientes herramientas:

- Eclipse como IDE
- JUnit para para el testing
- GitHub para el control de las versiones
- MYSQL de base de datos relacional
- MongoDB de base de datos no relacional
- Tomcat 8 de servidor de la aplicación
- Prime Facelets y JSF para el Frontend
- Para la persistencia de objetos en MYSQL usamos Hibernate(ORM)
- Para la persistencia de objetos en MongoDB usamos Morphia(ODM)
- Para el control de procesos
- Rollback y anidado de procesos utilizamos Quartz.

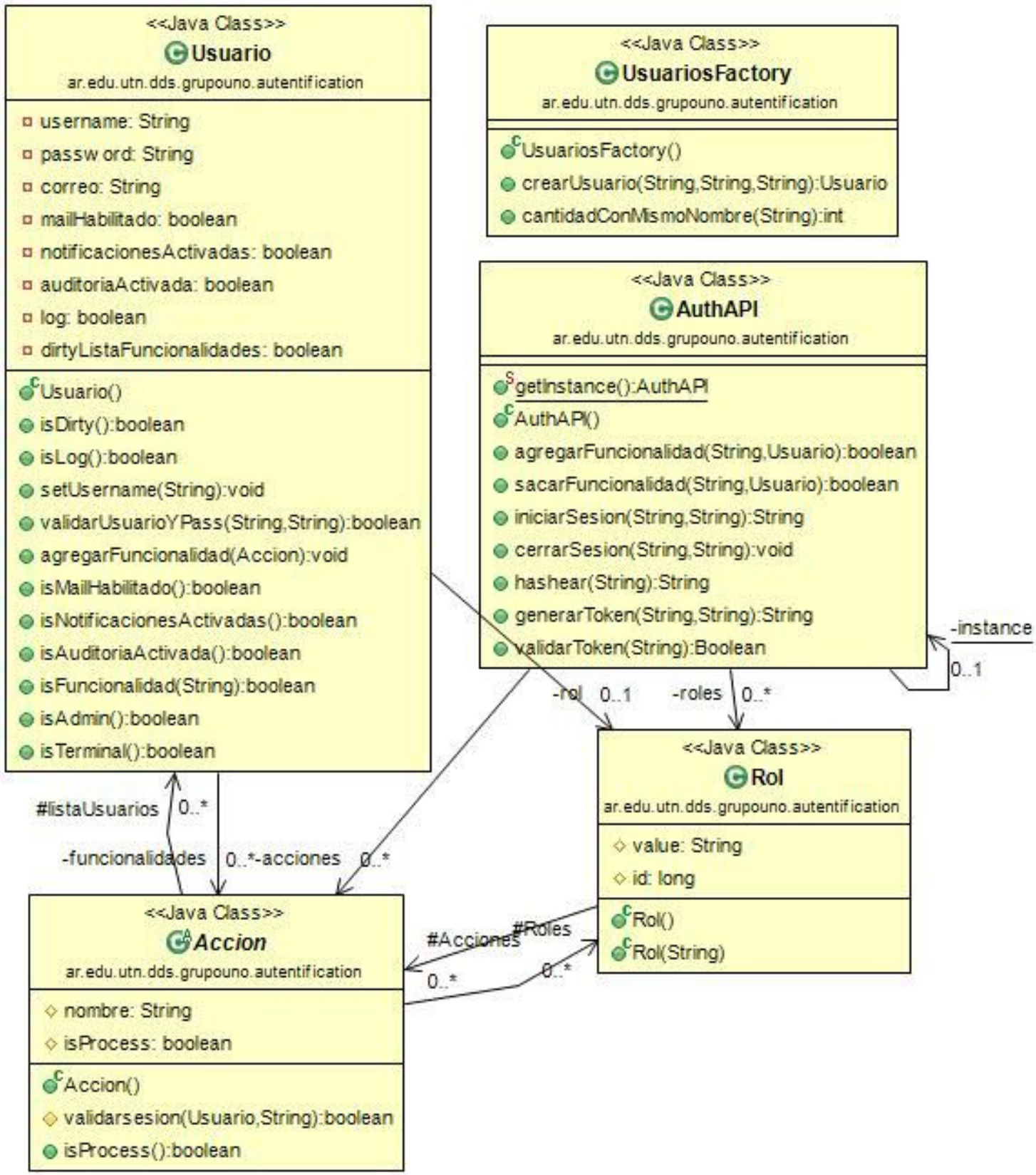
En MongoDB se persisten registros históricos (en cumplimiento con la entrega 7) y también se utiliza como caché de los servicios externos (en cumplimiento con la entrega 7),
en MYSQL se persistieron el resto de los objetos.
Las pruebas fueron realizadas en Eclipse Mars, Java 8,Tomcat 8.0.41, MYSQL 5.7, MongoDB 3.2.11.

Diagramas de clases de paquetes importantes

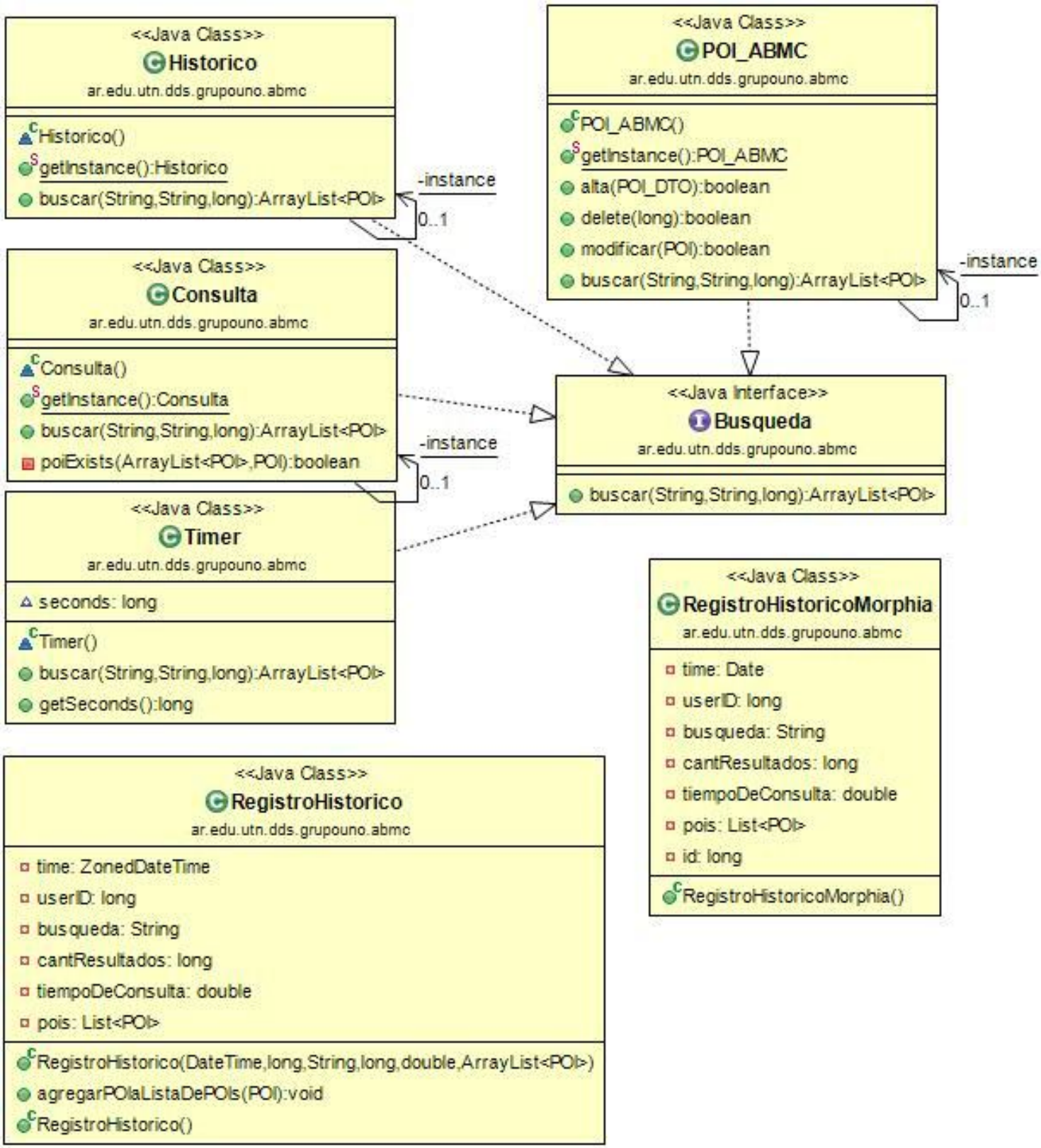
abmc.poi



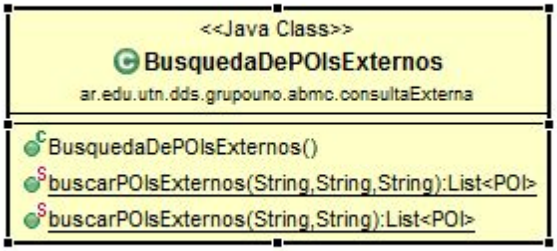
autentificación



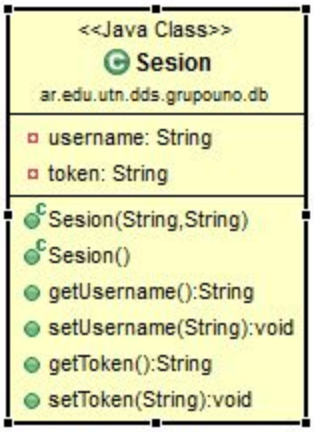
abmc


<<Java Class>>
Historico
ar.edu.utn.dds.grupouno.abmc
Historico()
getInstance():Historico
buscar(String,String,long):ArrayList<POI>
-instance
0..1
<<Java Class>>
POI_ABMC
ar.edu.utn.dds.grupouno.abmc
POI_ABMC()
getInstance():POI_ABMC
alta(POI_DTO):boolean
delete(long):boolean
modificar(POI):boolean
buscar(String,String,long):ArrayList<POI>
-instance
0..1
<<Java Class>>
Consulta
ar.edu.utn.dds.grupouno.abmc
Consulta()
getInstance():Consulta
buscar(String,String,long):ArrayList<POI>
poiExists(ArrayList<POI>,POI):boolean
-instance
0..1
<<Java Interface>>
Busqueda
ar.edu.utn.dds.grupouno.abmc
buscar(String,String,long):ArrayList<POI>
<<Java Class>>
Timer
ar.edu.utn.dds.grupouno.abmc
seconds: long
Timer()
buscar(String,String,long):ArrayList<POI>
getSeconds():long
<<Java Class>>
RegistroHistoricoMorphia
ar.edu.utn.dds.grupouno.abmc
time: Date
userID: long
busqueda: String
cantResultados: long
tiempoDeConsulta: double
pois: List<POI>
id: long
RegistroHistoricoMorphia()
<<Java Class>>
RegistroHistorico
ar.edu.utn.dds.grupouno.abmc
time: ZonedDateTime
userID: long
busqueda: String
cantResultados: long
tiempoDeConsulta: double
pois: List<POI>
RegistroHistorico(DateTime,long,String,long,double,ArrayList<POI>)
agregarPOIaListaDePOIs(POI):void
RegistroHistorico()

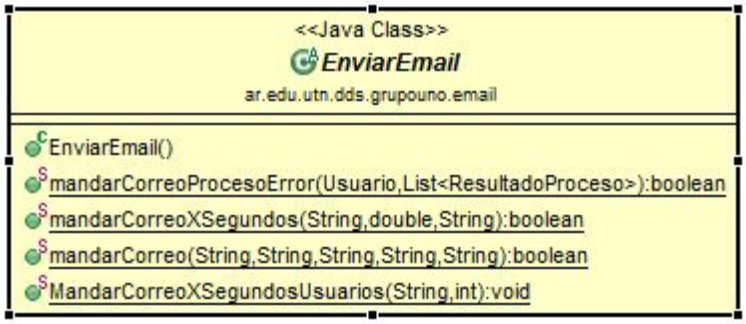
abmc.consultaExterna



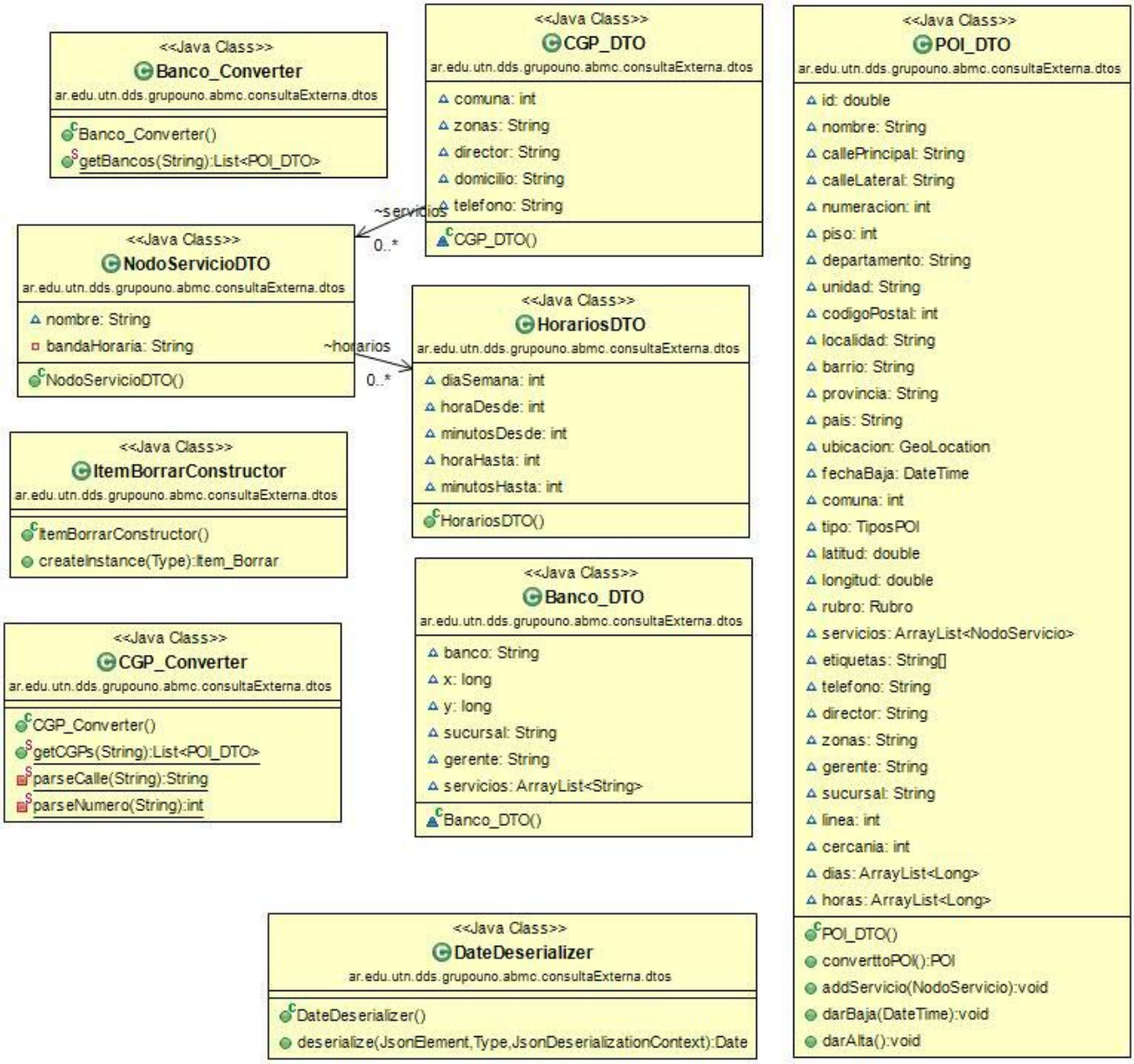
db



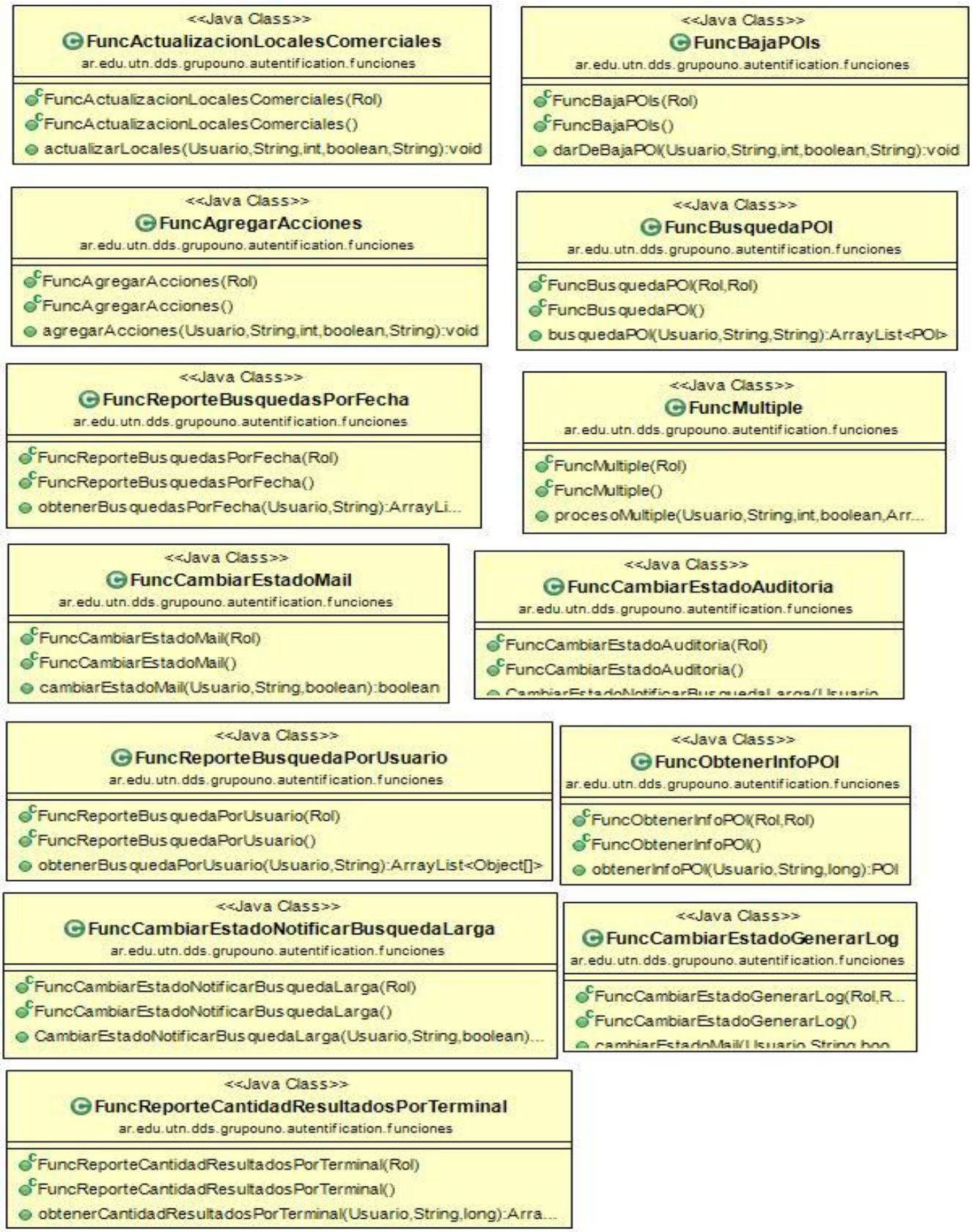
email



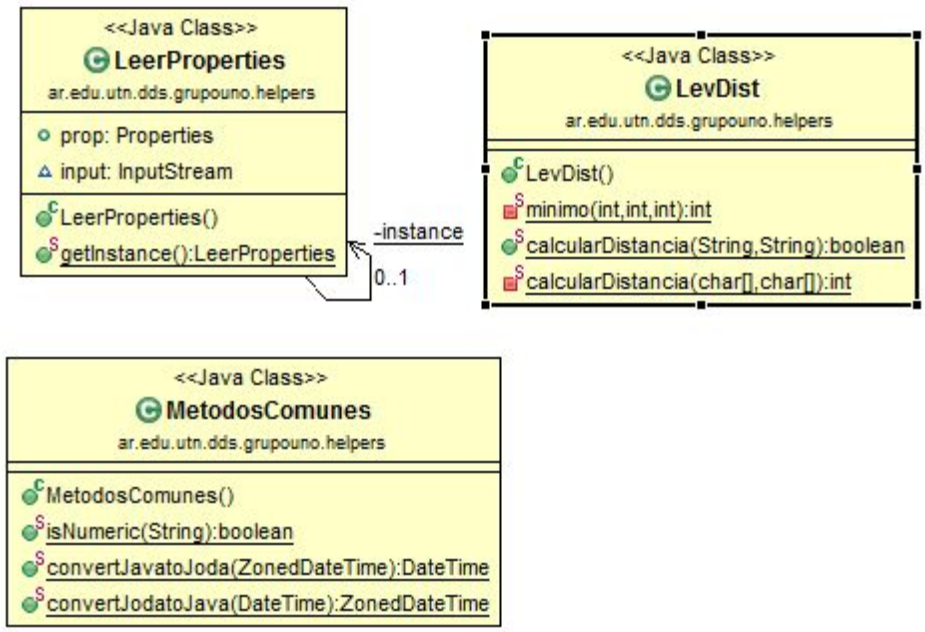
abmc.consultaExterna.dtos



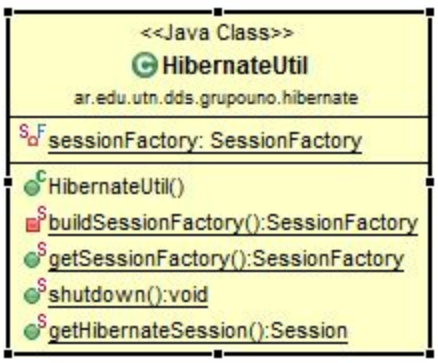
autentification.funciones



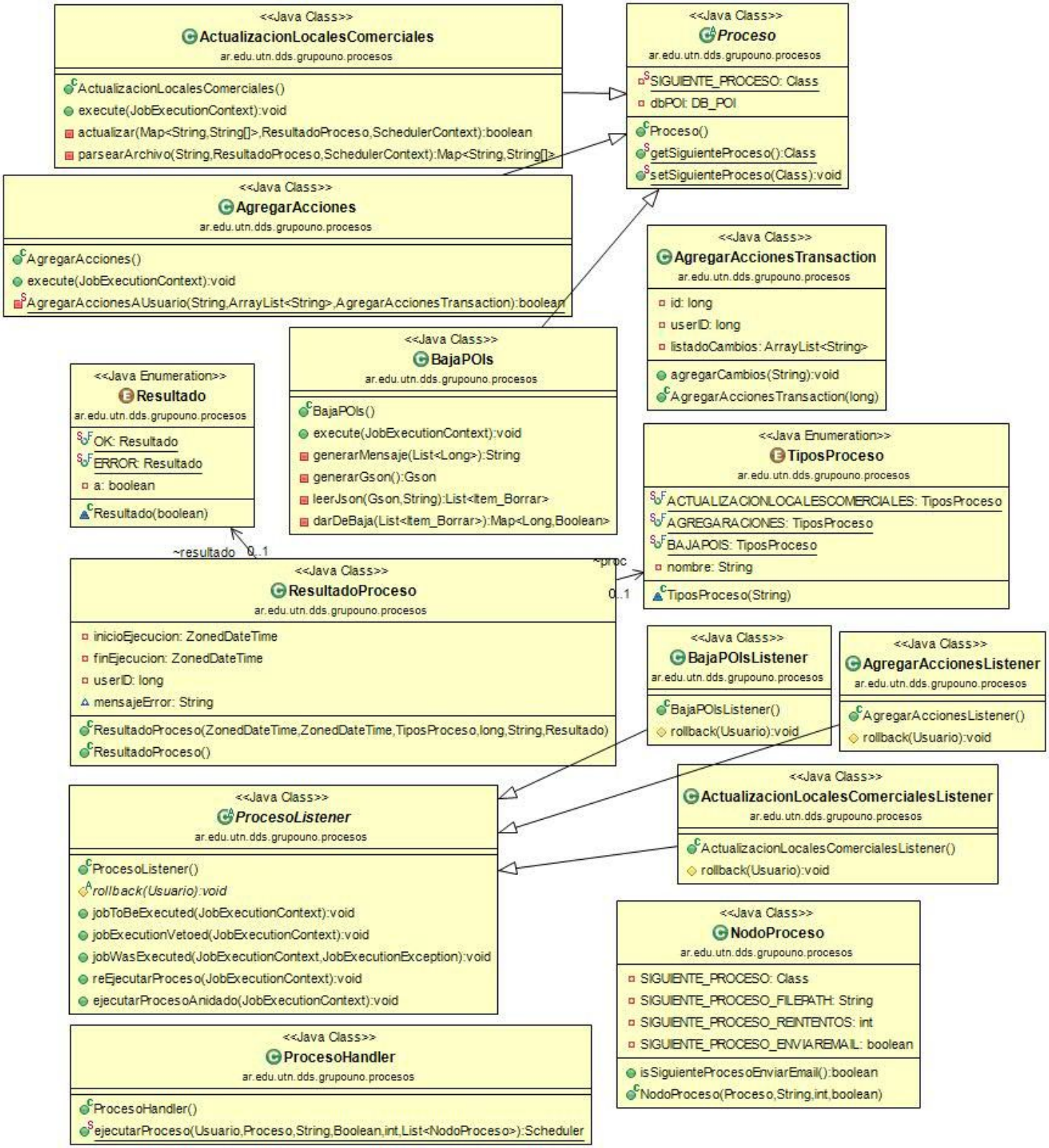
helpers



hibernate



procesos


<<Java Class>>
ActualizacionLocalesComerciales
ar.edu.utn.dds.grupouno.procesos
ActualizacionLocalesComerciales()
execute(JobExecutionContext):void
actualizar(Map<String,String[]>,ResultadoProceso,SchedulerContext):boolean
parsearArchivo(String,ResultadoProceso,SchedulerContext):Map<String,String[]>
<<Java Class>>
Proceso
ar.edu.utn.dds.grupouno.procesos
SIGUIENTE_PROCESO: Class
dbPOI: DB_POI
Proceso()
getSiguienteProceso():Class
setSiguienteProceso(Class):void
<<Java Class>>
AgregarAcciones
ar.edu.utn.dds.grupouno.procesos
AgregarAcciones()
execute(JobExecutionContext):void
AgregarAccionesAUsuario(String,ArrayList<String>,AgregarAccionesTransaction):boolean
<<Java Class>>
AgregarAccionesTransaction
ar.edu.utn.dds.grupouno.procesos
id: long
userID: long
listadoCambios: ArrayList<String>
agregarCambios(String):void
AgregarAccionesTransaction(long)
<<Java Class>>
BajaPOIs
ar.edu.utn.dds.grupouno.procesos
BajaPOIs()
execute(JobExecutionContext):void
generarMensaje(List<Long>):String
generarGson():Gson
leerJson(Gson,String):List<Item_Borrar>
darDeBaja(List<Item_Borrar>):Map<Long,Boolean>
<<Java Enumeration>>
Resultado
ar.edu.utn.dds.grupouno.procesos
OK: Resultado
ERROR: Resultado
a: boolean
Resultado(boolean)
~resultado
0..1
<<Java Enumeration>>
TiposProceso
ar.edu.utn.dds.grupouno.procesos
ACTUALIZACIONLOCALESCOMERCIALES: TiposProceso
AGREGARACIONES: TiposProceso
BAJAPOIS: TiposProceso
nombre: String
TiposProceso(String)
~proc
0..1
<<Java Class>>
ResultadoProceso
ar.edu.utn.dds.grupouno.procesos
inicioEjecucion: ZonedDateTime
finEjecucion: ZonedDateTime
userID: long
mensajeError: String
ResultadoProceso(ZonedDateTime,ZonedDateTime,TiposProceso,long,String,Resultado)
ResultadoProceso()
<<Java Class>>
BajaPOIsListener
ar.edu.utn.dds.grupouno.procesos
BajaPOIsListener()
rollback(Usuario):void
<<Java Class>>
AgregarAccionesListener
ar.edu.utn.dds.grupouno.procesos
AgregarAccionesListener()
rollback(Usuario):void
<<Java Class>>
ProcesoListener
ar.edu.utn.dds.grupouno.procesos
ProcesoListener()
rollback(Usuario):void
jobToBeExecuted(JobExecutionContext):void
jobExecutionVetoed(JobExecutionContext):void
jobWasExecuted(JobExecutionContext,JobExecutionException):void
reEjecutarProceso(JobExecutionContext):void
ejecutarProcesoAnidado(JobExecutionContext):void
<<Java Class>>
ActualizacionLocalesComercialesListener
ar.edu.utn.dds.grupouno.procesos
ActualizacionLocalesComercialesListener()
rollback(Usuario):void
<<Java Class>>
NodoProceso
ar.edu.utn.dds.grupouno.procesos
SIGUIENTE_PROCESO: Class
SIGUIENTE_PROCESO_FILEPATH: String
SIGUIENTE_PROCESO_REINTENTOS: int
SIGUIENTE_PROCESO_ENVIAREMAIL: boolean
isSiguienteProcesoEnviarEmail():boolean
NodoProceso(Proceso,String,int,boolean)
<<Java Class>>
ProcesoHandler
ar.edu.utn.dds.grupouno.procesos
ProcesoHandler()
ejecutarProceso(Usuario,Proceso,String,Boolean,int,List<NodoProceso>):Scheduler

repositorio

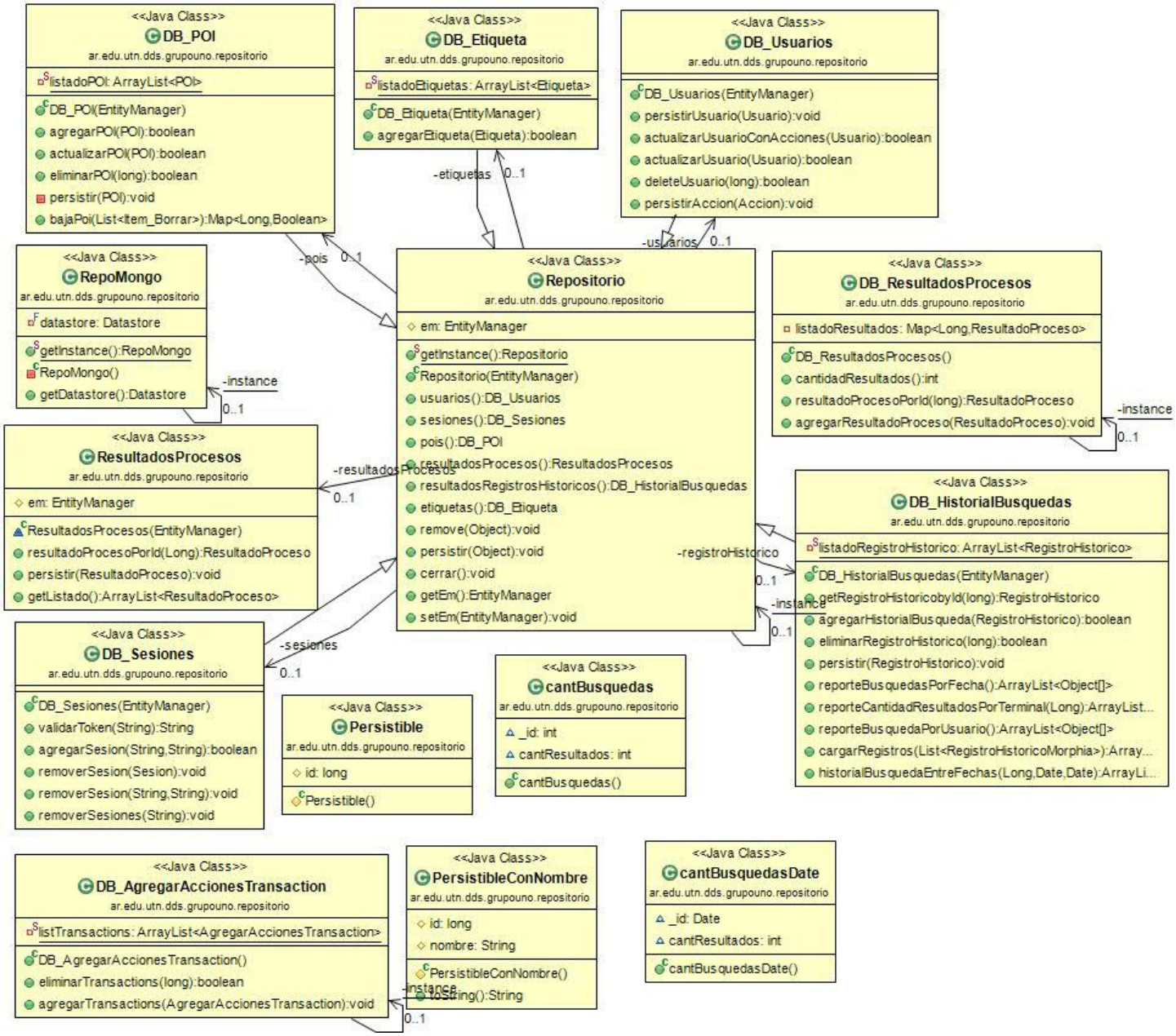


Diagrama de Entidad Relación de la Base de Datos en MYSQL