**Lineamientos generales desarrollo**

**en Java EE para CGN**

Versión 1.2

Abril/2017

# Propósito

El presente documento describe lineamientos en general a aplicar en CGN para el desarrollo de sistemas sobre la plataforma java (Java EE).

# Herramientas

Se utiliza como IDE para el desarrollo a Eclipse (en su última versión, actualmente Mars.2) sobre Java 8. Los fuentes se almacenan y comparten entre el equipo de desarrollo mediante un servidor subversion.

Como builder e integrador de proyectos se utiliza Ant (ya incorporado en versión Mars.2 de Eclipse) para el caso de proyectos que requieren poca o nula integración y Maven para casos de integración más complejos.

Los servidores de aplicación utilizados en todos los ambientes son Jboss EAP (última versión 6.4).

Las bases de datos son Oracle Server (versión 10 para el caso del SIIF, versión 12c para otros proyectos). Para el modelado de datos (tablas, vistas, claves, etc) se recomienda SQL Developer ya que está orientado a bases Oracle y es de uso libre.

Se armarán tests de pruebas de las clases desarrolladas usando JUnit (provisto también por Eclipse) como modo de automatizar los tests ante cada nueva versión del sistema desarrollado; para el caso de Web Service se armarán utilizando SOAP UI.

# Arquitectura

La arquitectura emplea los elementos estándares de Java EE en sus últimas versiones. En la medida de lo posible se evitarán frameworks extra, tales como spring o apache axis para evitar dependencias a futuro y tiempos de aprendizaje de las herramientas adicionales. En caso de querer emplearlos de todos modos, se debe justificar su uso y debe ser aprobado por el equipo de arquitectura de la CGN.

Debido a la diversidad de opciones, se presentan los lineamientos generales a utilizar, según los grupos/capas:

* Presentación:
* Páginas estáticas en los estándares HTML/CSS y javascript
* Páginas dinámicas con JSP y JSF. Para interacciones más complejas con el usuario (UI) utilizar PrimeFaces. Para sistemas con mucha interacción (gran cantidad de pantallas y/o complejidad de las mismas) puede utilizarse a AngularJS.
* Negocio: EJB: session bean en general y message driven beans para manejo de eventos asincrónicos (por ejemplo, para espera de mails o archivos con mensajes). Como alternativa a EJB la capa de negocio puede implementar Web Services para ser consumidos por la Presentación.
* Integración:
  + acceso a base de datos:
    - JDBC en general
    - JPA (con la implementación EclipseLink) para interacción con una tabla de muchas columnas, o bien para sistemas que necesiten el acceso por separado a varias tablas. Se evita así tener que implementar muchas conversiones entre tipos de datos (la relación entre tabla y objeto java).
    - Paquete pl/sql para procesamiento más complejo de tablas: por temas de performance se pueden desarrollar paquetes pl/sql en vez de varias sentencias sql con JDBC.
  + web services: tipo SOAP mediante JAX-WS, tanto como servidor como para consumo de web services. En caso requerirse consumir un web service REST utilizar JAX-RS.

# Buenas prácticas

Para facilitar el trabajo en equipo, producir código legible y facilitar el mantenimiento posterior, se sugieren “buenas prácticas” a aplicar en el desarrollo de los sistemas en Java EE.

## Log de Aplicación

Cada aplicación debe generar su propio log de aplicación, para lo cual se utiliza la librería **log4j** (versión 1.2.17 o posterior). Se puede definir la configuración de logueo en el archivo de propiedades “log4j.properties” dentro de la aplicación o desde la administración de JBoss, de forma tal que la aplicación siempre genere su propio log.

## Testing con JUnit o SOAP UI

Desarrollar casos de prueba repetibles con JUnit o SOAP UI (para el caso de Web Services) de modo de poder ejecutarlos en su totalidad en el momento de verificar una modificación a un sistema. Esto permite reducir los tiempos de test y automatizar la mayoría de las pruebas.

Los casos de prueba deben abarcar al menos:

* cada funcionalidad significativa del sistema
* casos de borde
* al menos un caso “normal”
* casos de error

## Codificación

Recomendaciones sobre la buena codificación de los programas.

**Comentarios**

En cada clase indicar una descripción de la misma, fecha y autor. Ídem para cada modificación a la misma establecer la razón del cambio, junto a fecha y autor.

Dentro de cada clase comentar lo que ayude a comprender su funcionamiento.

**Nomenclatura**

Para nombres de identificadores en general usar el “CamelCase”, esto es, nombres compuestos en donde la primera letra de cada palabra está en mayúsculas. Para:

* clases: la primera letra va en mayúsculas y suele ser un sustantivo.
* métodos: la primera letra en minúsculas. Generalmente es un verbo.
* variables: primera letra en minúsculas. Usar en lo posible nombres cortos.
* constantes: no se emplea CamelCase. Usar todas las letras en mayúsculas y separar las palabras por “\_”.

**Mantenibilidad**

Recomendaciones generales de programación para facilitar el mantenimiento del código:

* Métodos y constructores sencillos: con pocas líneas de código, parámetros y pocos puntos de decisión
* Escribir código una vez: evitar lógica duplicada
* Separar temas en clases: no escribir clases complejas, crear una clase por cada “tema” o elemento de interés.