

2016

MINISTERIO PÚBLICO
FISCALÍA DE LA NACIÓN

*Promoviendo
una cultura de paz*

GUIA DE DESARROLLO

Versión 1.0

Identificación del Documento

Tipo de documento:

Documentos de Arquitectura

| | | |
|--------------------------|-------------------|-----------------------------------|
| Código Documento: | Revisión: | N° Total de páginas: |
| Developer Guide | 1 | 56 |
| Fecha Creación: | 13-05-2016 | Fecha de Modificación: |
| Elaborado por: | | |
| Proveedor | | |
| Aprobado por: | | |

Ciclo de Aprobación

| Rol | Nombre | Firma | Fecha |
|--------------------|---------------|--------------|----------------|
| Creado por: | Raúl Caso | | 13-05- 2016 |

Historia de Cambios

| Versión (Estado) | Autor | Descripción del Cambio | Fecha |
|-----------------------------|--------------|-------------------------------|----------------|
| 1 | Raúl Caso | Creación del documento | 13-05- 2016 |

Tabla de Contenido

| | |
|---|----|
| 1. INTRODUCCIÓN | 8 |
| 1.1 Propósito..... | 8 |
| 1.2 Alcance | 9 |
| 1.3 Definiciones, Siglas y Abreviaturas | 9 |
| 1.4 Audiencia del Documento de Arquitectura. | 10 |
| 2. Resumen Estándares de Desarrollo..... | 11 |
| 3. Ámbito de la Arquitectura | 13 |
| 3.1 Descripción de Tecnologías y Frameworks..... | 13 |
| 3.2 Herramientas..... | 14 |
| 4. Arquitectura de los Aplicativos | 15 |
| 4.1 Estructura de los Proyectos | 15 |
| 4.2 MPFN-SECURITY..... | 15 |
| 4.3 MPFN-CONFIGURATION | 16 |
| 4.4 MPFN-ENTITYMANAGER | 16 |
| 4.5 MPFN-CORE | 16 |
| 4.6 MPFN-JSF-CORE | 16 |
| 5. Patrón de Diseño ECB | 17 |
| 5.1 Boundary..... | 17 |
| 5.2 Controller..... | 18 |
| 5.3 Entity..... | 19 |
| 6. Nomenclatura en los Proyectos | 21 |
| 6.1 Paquetes Boundary | 21 |
| 6.2 Paquetes Controller | 21 |
| 6.3 Paquetes Entity | 21 |
| 6.4 Paquetes Type | 21 |
| 6.5 Beans JSF | 22 |
| 6.6 Beans Boundary | 22 |
| 6.7 Beans Controller | 22 |
| 6.8 Archivos Properties..... | 22 |
| 7. Componentes de la Arquitectura | 23 |
| 7.1 Capa de Presentación..... | 23 |

| | | |
|-------|---|----|
| 7.1.1 | JSF | 23 |
| 7.2 | Seguridad de la Capa de Negocio | 23 |
| 7.3 | Utilización del CrudServiceController | 24 |
| 7.3.1 | Configuración | 25 |
| 7.3.2 | Utilización..... | 25 |
| 7.3.3 | Auditoria | 26 |
| 7.4 | Mapeo utilizando JPA | 27 |
| 7.4.1 | Auditoria | 27 |
| 7.4.2 | Concurrencia..... | 28 |
| 7.4.3 | Tipos de datos LocalDateTime y LocalDate | 29 |
| 7.4.4 | Bean Validation..... | 29 |
| 7.5 | Logging..... | 31 |
| 7.6 | Gestión de Errores Capa Negocio | 31 |
| 7.7 | Configuración de Parámetros de Despliegue | 32 |
| 8. | Configuraciones de Herramientas para Desarrollo | 34 |
| 8.1 | JDK 8 | 34 |
| 8.2 | Wildfly 10..... | 34 |
| 8.2.1 | Instalación | 34 |
| 8.2.2 | Configuración Datasource..... | 36 |
| 8.3 | Eclipse Mars | 37 |
| 8.3.1 | Configuración Proxy | 37 |
| 8.3.2 | Configuración JDK..... | 38 |
| 8.3.3 | Jboss Tools | 39 |
| 8.3.4 | Subclipse | 40 |
| 8.3.5 | Jautodoc..... | 40 |
| 8.3.6 | Configurar Maven settings.xml | 41 |
| 8.3.7 | Configuración de Repositorio de Archetypes..... | 43 |
| 8.3.8 | Configuración del Jautodoc | 44 |
| 9. | Pasos para el Desarrollo | 45 |
| 9.1 | Acceso al Código de los Proyectos | 45 |
| 9.1.1 | Agregar Repositorio SVN | 45 |
| 9.1.2 | Checkout del Código..... | 46 |
| 9.1.3 | Convertir a Maven Project | 47 |

| | | |
|-------|--|----|
| 9.2 | Crear nuevos Proyectos..... | 48 |
| 9.2.1 | Crea Nuevo Proyecto Back End..... | 48 |
| 9.3 | Documentación de Clases | 51 |
| 9.3.1 | JAutodoc | 51 |
| 9.4 | Trabajar con SVN en los Proyectos..... | 53 |
| 9.4.1 | Subir un proyecto nuevo al SVN..... | 53 |
| 9.4.2 | Subir Código al SVN | 56 |
| 9.4.3 | Actualizar Código del SVN | 58 |

Índice de Figuras

| | |
|---|----|
| Ilustración 1 Módulos de la aplicación | 15 |
| Ilustración 2 Diagrama de patrón | 17 |
| Ilustración 3 Ejemplo de un Boundary | 18 |
| Ilustración 4 Implementación con CDI..... | 18 |
| Ilustración 5 Implementación como EJB | 19 |
| Ilustración 6 Mapeo de entidad | 19 |
| Ilustración 7 Utilización de Version y LocalDateTime..... | 20 |
| Ilustración 8 Implementando Auditable para campos de auditoria | 20 |
| Ilustración 9 Interface a implementar | 27 |
| Ilustración 10 Tabla con la lista de anotaciones para validaciones | 30 |
| Ilustración 11 Mapeador de excepciones RestFul | 32 |
| Ilustración 12 Ejemplo de definición de parámetros..... | 32 |
| Ilustración 13 Ejemplo de utilización en el código | 33 |
| Ilustración 14 Descomprimir en carpeta sin espacios en blanco en los nombres..... | 34 |
| Ilustración 15 Creación usuario administrador | 35 |
| Ilustración 16 Iniciando por consola | 35 |
| Ilustración 17 Ingresar usuario y password creados | 35 |
| Ilustración 18 Consola administrativa | 36 |
| Ilustración 19 Copiar a la carpeta modules del wildfly | 36 |
| Ilustración 20 Agregar el driver a la configuración..... | 36 |
| Ilustración 21 Atributos de datasources..... | 37 |
| Ilustración 22 Configurar proxy | 38 |
| Ilustración 23 Configuración utilización del JDK..... | 38 |
| Ilustración 24 Eclipse Marketplace..... | 39 |
| Ilustración 25 Instalar Jboss tools..... | 39 |
| Ilustración 26 Instalar Subclipse | 40 |
| Ilustración 27 Instalar Jautodoc..... | 40 |
| Ilustración 28 Configurar archivo settings | 41 |
| Ilustración 29 Agregar repositorio archetype | 43 |
| Ilustración 30 Repositorio agregado | 43 |
| Ilustración 31 Configurar Plantilla | 44 |
| Ilustración 32 Agregar Repositorio | 45 |
| Ilustración 33 Checkout código..... | 46 |
| Ilustración 34 Check out in workspace | 46 |
| Ilustración 35 Convertir a Maven..... | 47 |
| Ilustración 36 Proyecto reconocido como Maven | 48 |
| Ilustración 37 Click nuevo proyecto | 48 |
| Ilustración 38 Utilizar archetypes..... | 49 |
| Ilustración 39 Seleccionar el archetype..... | 49 |
| Ilustración 40 Configuración nombre de proyectos..... | 50 |
| Ilustración 41 Proyecto Creado..... | 50 |
| Ilustración 42 Crear nueva clase | 51 |

| | |
|---|----|
| Ilustración 43 Agregar Comentarios | 52 |
| Ilustración 44 Cabecera estándar de comentario..... | 52 |
| Ilustración 45 Compartir proyecto | 53 |
| Ilustración 46 Seleccionar SVN | 53 |
| Ilustración 47 Configurar nuevo repositorio | 54 |
| Ilustración 48 Repositorio donde se subirá el código | 54 |
| Ilustración 49 Nombre de proyecto | 54 |
| Ilustración 50 Añadir al repositorio local | 55 |
| Ilustración 51 Seleccionar archivos para commit | 55 |
| Ilustración 52 Confirmación y comentario del commit | 56 |
| Ilustración 53 Número de versión de archivos | 56 |
| Ilustración 54 Commit modificaciones | 57 |
| Ilustración 55 Especificar el motivo de la modificación y confirmar | 57 |
| Ilustración 56 Nueva versión después del commit..... | 57 |
| Ilustración 57 Configuración de conflictos..... | 58 |
| Ilustración 58 Actualizar porción del código | 58 |

1. INTRODUCCIÓN

El presente documento corresponde a la definición de una guía de desarrollo para los aplicativos en Java basados en la nueva arquitectura, documento que centrará principalmente en la descripción de los componentes de la arquitectura y en cómo serán usados en la construcción e implementación de los sistemas.

1.1 Propósito

Presentar el diseño técnico de la arquitectura a utilizar para el desarrollo de los proyectos (capas de la arquitectura, componentes de servicios, mecanismo de integración, mecanismos de persistencia, mecanismo de manejo de errores etc.).

Asimismo en este documento se describen las herramientas y tecnologías a utilizar en la construcción, pruebas y despliegue de la solución.

1.2 Alcance

El presente documento muestra y describe las siguientes consideraciones

- a) Las herramientas que se emplearán para la construcción
- b) Patrones de diseños y frameworks a usar por cada capa de la aplicación, nomenclaturas.
- c) Mecanismo de Interface
- d) Mecanismo de persistencia y registro de la información en base de datos

1.3 Definiciones, Siglas y Abreviaturas

Las principales siglas utilizadas a lo largo de este documento, son:

- Java EE: Java Enterprise Edition
- JVM: Java Virtual Machine
- TO: Transfer Object
- EJB: Enterprise Java Bean
- JPA: Java Persistence API
- POJO: Plain Old Java Object
- JMS: Java Message Service
- LDAP: Lightweight Directory Access Protocol
- JAX-RS: Java API para RestFul Web Services
- JSF: Java Server Faces
- CDI: Context Dependency Injection
- JWT: Json web Token

1.4 Audiencia del Documento de Arquitectura.

La audiencia objetivo del presente documento corresponde a:

- a) EL EQUIPO DE ARQUITECTURA el cual utilizará este documento para seguir las pautas y directivas de implementación del Software, respetando que se cumplan con las convenciones aquí establecidas.
- b) EQUIPO DESARROLLO, los cuales encontrarán en este documento el detalle de la documentación de los bloques básicos y componentes, en los que se construirá gran parte de los elementos del sistema.
- c) EQUIPO MANTENIMIENTO, los cuales encontrarán en este documento los lineamientos base para la construcción de los componentes de software a ser mantenidos.

2. Resumen Estándares de Desarrollo

Se procede a listar un resumen de las reglas de programación que se deben respetar para el desarrollo de las aplicaciones.

- El manejo de acciones Ajax siempre se debe utilizar los atributos **process** para especificar los componentes a enviar en la operación POST y **update** para especificar solo los componentes que tienen que ser actualizados.
- En la inclusión de componentes Dialogs, Helpers, etc estos no deben cargarse al momento de mostrar la página principal sino solo cuando sean activados(al presionar botones o links que realmente muestran esas páginas)
- Verificar en cada pantalla que el uso de la tecla TAB funcione correctamente, si no es así utilizar tab index en los componentes para ordenar el flujo de componentes.
- Al momento de utilizar el boto de Regresar en una pantalla que tiene como página anterior una búsqueda los resultados deben mantenerse en la pantalla.
- Los Enums(Types) solo deben ser usados para comparar los valores numéricos.
- Los títulos de las páginas deben estar centralizadas en el archivo Messages_es.properties.
- Todos los componentes JSF tienen que tener un identificador que tenga un valor representativo al componente que represente(osea no usar val1 ,val2, etc o cosas parecidas)
- Como regla todas las validaciones realizadas individualmente en campos de un formulario deben ser validadas de nuevo en la capa de negocio para asegurar la consistencia de los datos.
- Solo lógica de presentación deberá ser realizada en la capa JSF, cualquier regla de negocio deberá ser realizada en la capa Boundary.
- Las entidades por default deben tener mapeadas asociaciones con el atributo Lazy.
- Es obligatorio que comenten la firma de los métodos (métodos en los Boundary o Controller) y cualquier porción de código que tenga una lógica de complejidad media a alta explicando brevemente la funcionalidad que realiza.
- Usar Criteria solo para queries simples, cuando la query es compleja se deberá realizar con JPQL o Nativo para facilitar el mantenimiento.

- Se recomienda que para wars con varios módulos las páginas xhtml deberá seguir la siguiente estructura
 - paginas/nombreModulo/nombreOpcion/search + "nombreOpcion" + .xhtml
 - paginas/nombreModulo/nombreOpcion/register + "nombreOpcion" + .xhtml
 - paginas/nombreModulo/nombreOpcion/view + "nombreOpcion" + .xhtml

3. Ámbito de la Arquitectura

3.1 Descripción de Tecnologías y Frameworks

A continuación se proporcionan una breve descripción de las tecnologías usadas.

| Capas de la Arquitectura | Descripción | Tecnologías, propuestas | Patrones del Software (En caso sean Utilizados) | Software/Herramientas |
|--|--|--|---|---|
| Capa de Presentación (Capa Browser) | Son Interfaces gráficas en la cual los usuarios interactuarán con el sistema. | JSF 2.x , Primefaces 5.x/ Angular.js 2.x | MVC | Facelet, CSS y Entorno de desarrollo Eclipse. |
| Capa de Integración de Servicios | Esta capa proporciona interfaz a todos los componentes disponibles tratándolos como servicios reutilizables | Rest Services | Patrón Boundary | JAX-RS |
| Capa de Negocio | Esta capa proporciona la lógica real del negocio que actúa como el mediador entre la persistencia y la capa del Componente | Stateles EJB, CDI beans | Patrón IOC | CDI 1.1, EJB 3.2 |
| Capa de acceso a Datos | Esta capa proporciona persistencia del objeto, consiste en el mapeo de la relación Objeto Java y Objeto de Base de Datos. | JPA 2.1 con Hibernate | Patrón DAO | JPA 2.1 |

Tabla 1 Tabla resumen de tecnologías usadas

3.2 Herramientas

Se listan las herramientas que serán utilizadas como parte del desarrollo y pruebas de la implementación.

| Tipo de Herramienta | Nombre y Versión |
|--|---|
| Java JDK | Java JDK 1.8.077 o superior 64bits. |
| Servidor de Aplicaciones | Wildfly 10 <ul style="list-style-type: none"> Servidor de aplicaciones Java EE 7, con los servicios transversales propios de un contenedor con soporte de estándares bajo la especificación EJB 3.2, Servlet 3.1, JSF 2.2, CDI 1.1, JPA 2.1, JNDI 1.2, JMX 1.2, JTA 1.2, JMS 2.0, JAAS 1.2, etc. |
| Control de Versiones | SVN Tortoise 1.9 |
| Diseñador de Reportes | Herramienta para la creación de reportes. Se usara IReport 6 |
| Pruebas unitarias | Se usara Junit 4.10, se podrá usar versiones superiores que sean estables. |
| Pruebas Integrales de Modulo | Se usara Arquillian como framework en su versión 1.1.11.Final |
| Herramienta para la construcción y ejecución de tareas repetitivas | Maven 3.3 |
| Herramientas para acceso a Datos | DBArtisan |
| Herramienta de Desarrollo | Eclipse Mars con el plugin Jboss tools |

Tabla 2 Tabla de listado de herramientas usadas

4. Arquitectura de los Aplicativos

Los aplicativos Web deberán estar conformados por 2 proyectos, un proyecto front end (JSF/Angular) y un proyecto Back End (Java RestFul) de esta manera al tener desacoplados la interfaz de usuario de la lógica de negocio es más fácil hacer escalar la aplicación en un sistema de balanceo de solo nodos back end donde estará implementada la lógica de negocio.

Para los casos donde el aplicativo web requiera funcionalidad simple con una estimación de consumo de recursos bajos es válido que el front end y back end sean implementados en un mismo proyecto manteniendo las directrices de desarrollo establecidas facilitando así el refactoring en caso sea necesario la separación de las funcionalidad por motivo de mejorar el escalamiento.

4.1 Estructura de los Proyectos

Las aplicaciones están formadas por proyectos de uso común de tipo JAR y proyectos WARs para la implementación de los módulos de negocio.

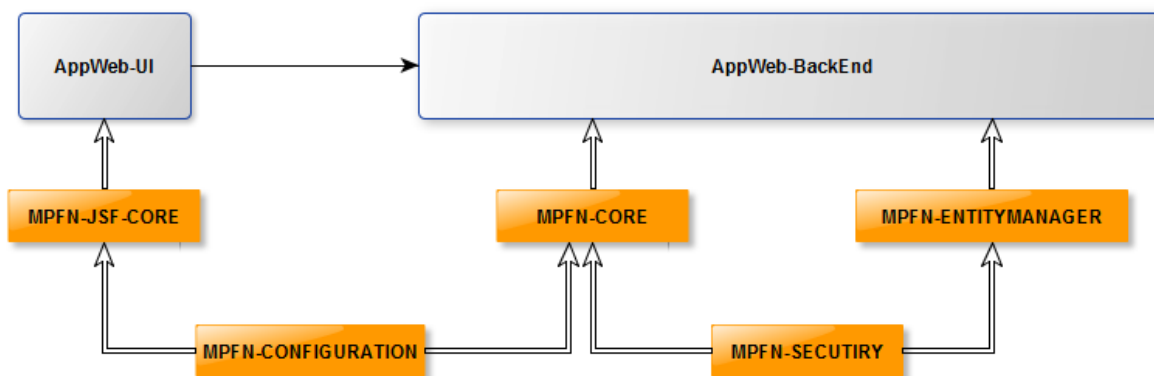


Ilustración 1 Módulos de la aplicación

4.2 MPFN-SECURITY

Proyecto que contiene los servicios de configuración y encriptación del token de seguridad usado para la validación de autenticación de los servicios de negocio expuestos. También contiene a la clase UserTrack que es utilizada para el envío de los datos del usuario que ejecuta los servicios.

4.3 MPFN-CONFIGURATION

Proyecto que implementa las siguientes funcionalidades:

- Encargado de la configuración de parámetros de la aplicación utilizando el archivo `ApplicationConfiguration.properties` y la anotación `@Configurable`
- Encargado de configurar el logging de la aplicación inyectando el componente `MpfnLogger`

4.4 MPFN-ENTITYMANAGER

Proyecto que implementa las siguientes funcionalidades:

- Encargado de manejar la abstracción y reutilización de métodos para realizar las operaciones CRUD en la base de datos utilizando JPA
- Encargado de registrar las auditorias de registros obtenidos en la capa de Negocio.
- Encargado de brindar el mapping de los nuevos tipos de datos `LocalDateTime` y `LocalDate` a `Date` para poder ser utilizado en el mapeo de JPA.
- Encargado de recibir la configuración paramétrica del nombre de la unidad de persistencia

4.5 MPFN-CORE

Proyecto que implementa las siguientes funcionalidades:

- Encargado del manejo de excepciones y su conversión a códigos de error HTTP.
- Encargado de verificar la seguridad de acceso a los servicios Restful configurados con la anotación `@TokenAuthenticated`.

4.6 MPFN-JSF-CORE

Proyecto que implementa las siguientes funcionalidades:

- Encargado de la configuración de parámetros comunes de JSF
- Encargado de las configuración de parámetros comunes en el `web.xml`
- Encargado de crear las clases `Base` y `Scopes` estándares que se utilizaran.
- Encargado brindar los templates comunes para la utilización de layouts uniformes en las aplicaciones

5. Patrón de Diseño ECB

Algunas veces llamado BCE (Boundary, Controller, Entity)

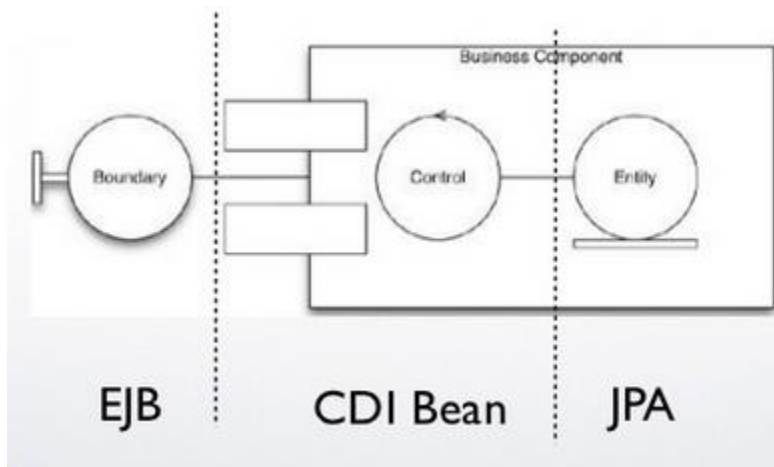


Ilustración 2 Diagrama de patrón

El patrón de diseño utilizado para desarrollar aplicativos en la arquitectura será el ECB, el cual distribuye la responsabilidad de los componentes en 3 elementos agrupados en paquetes:

5.1 Boundary

Encargado de ser el único punto de acceso de la aplicación con cualquier sistema/actor externo, implementado como un componente EJB Stateless que implementa anotaciones Restful.

Cada método del Boundary debe ser transaccional (comportamiento por defecto en los EJB) y siguiendo el patrón de nomenclatura RestFul el nombre del componente deberá terminar con la palabra “Resource”.

La ruta de acceso a un componente boundary sigue el siguiente patrón:

`http(s)://rutaservidor(:8080)/nombreapp/resources/(uri rest)`

Es decir ya están configurados para mapear los servicios Rest después del path “resources”

```
@Path("aplicaciones")
@Stateless
public class AplicacionResource {

    /** La crud service. */
    @EJB
    CrudServiceController crudService;

    /**
     * Crear aplicacion.
     *
     * @param aplicacion el aplicacion
     * @return the response
     */
    @POST
    @Consumes(MediaType.APPLICATION_JSON)
    @Produces(MediaType.APPLICATION_JSON)
    @TokenAuthenticated
    @UserAudit
    public Response crearAplicacion(Aplicacion aplicacion) {
        aplicacion = crudService.create(aplicacion);
        return Response.ok(aplicacion.getIdAplicacionPk()).build();
    }
}
```

Ilustración 3 Ejemplo de un Boundary

5.2 Controller

Encargado de implementar la lógica de negocio que será accedida y orquestada por los componentes Boundary.

Normalmente implementados como componentes CDI que utilizan el scope `@ApplicationScoped` o `@RequestScoped`, en caso se necesite algunos servicios como ejecución asíncrona (`@Asynchronous`) o ejecutar alguna funcionalidad específica del `EntityManager` se podrá utilizar componentes EJB para su implementación.

Como estándar el nombre del componente debe terminar en “Controller”

```
@ApplicationScoped
public class EstadisticaController {

    /**
     * Instancia un nuevo estadistica controller.
     */
    public EstadisticaController() {
        // TODO Auto-generated constructor stub
    }
}
```

Ilustración 4 Implementación con CDI

```
@Stateless
public class ParametrosController extends CrudServiceController {

    /**
     * Default constructor.
     */
    public ParametrosController() {
        // TODO Auto-generated constructor stub
    }
}
```

Ilustración 5 Implementación como EJB

5.3 Entity

Encargado de mapear las entidades a las Tablas utilizando JPA 2.1, colocar validaciones utilizando Bean Validation (@NotNull, etc) controlando la concurrencia de modificaciones en las entidades utilizando @Version.

Las clases que representan a las entidades deben implementar los métodos de la interface Auditable para realizar la auditoria a nivel de registro en las entidades.

```
@Entity
@Table(name="aplicacion")
@NamedQuery(name="Aplicacion.findAll", query="SELECT a FROM Aplicacion a")
public class Aplicacion implements Serializable, Auditable {

    /** La Constante serialVersionUID. */
    private static final long serialVersionUID = 1L;

    public static final String LISTAR_APLICACIONES = "Aplicacion.findAll";

    /** La id aplicacion pk. */
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    @Column(name="id_aplicacion_pk")
    private Long idAplicacionPk;

    /** La nombre. */
    @NotNull
    private String nombre;
```

Ilustración 6 Mapeo de entidad

```
/** La fecha modificacion. */
@Column(name="fecha_modificacion")
private LocalDateTime fechaModificacion;

/** La ip modificacion. */
@Column(name="ip_modificacion")
private String ipModificacion;

/** La menu modificacion. */
@Column(name="menu_modificacion")
private String menuModificacion;

/** La version. */
@Version
private Long version;
```

Ilustración 7 Utilización de Version y LocalDateTime

```
/* (non-Javadoc)
 * @see pe.gob.mpfن.entitymanager.auditoria.Auditablen#getListForAudit()
 */
@Override
public Map<String, List<? extends Auditablen>> getListForAudit() {
    // TODO Auto-generated method stub
    return null;
}

/* (non-Javadoc)
 * @see pe.gob.mpfن.entitymanager.auditoria.Auditablen#setAudit(pe.gob.mpfن.entitymanager.audit
 */
@Override
public void setAudit(UserTrack userTrack) {
    this.fechaModificacion=userTrack.getAuditTime();
    this.usuarioModificacion=userTrack.getUserName();
    this.ipModificacion=userTrack.getIpAddress();
}
```

Ilustración 8 Implementando Auditable para campos de auditoria

6. Nomenclatura en los Proyectos

6.1 Paquetes Boundary

Deben ser nombrados en minúscula, singulares y tener un nombre que represente el módulo de negocio y responsabilidad que representa.

| Modulo | Ejemplo |
|--------------|---|
| Batch | pe.gob.mpfm.framework.batchprocess.boundary |
| Notificacion | pe.gob.mpfm.notification.boundary |
| Eventos | pe.gob.mpfm.httpevent.publish.boundary |

6.2 Paquetes Controller

Deben ser nombrados en minúscula, singulares y tener un nombre que represente el módulo de negocio y responsabilidad que representa.

| Modulo | Ejemplo |
|--------------|---|
| Batch | pe.gob.mpfm.framework.batchprocess.controller |
| Notificacion | pe.gob.mpfm.notification.controller |
| Eventos | pe.gob.mpfm.httpevent.publish.controller |

6.3 Paquetes Entity

Deben ser nombrados en minúscula, singulares y tener un nombre que represente el módulo de negocio y responsabilidad que representa

| Modulo | Ejemplo |
|--------------|---|
| Batch | pe.gob.mpfm.framework.batchprocess.entity |
| Notificacion | pe.gob.mpfm.notification.entity |
| Eventos | pe.gob.mpfm.httpevent.publish.entity |

6.4 Paquetes Type

Relacionado con el paquete entity, permite agrupar los enums que contienen constantes de negocio relacionadas a las entidades

| Modulo | Ejemplo |
|--------------|--|
| Batch | pe.gob.mpfm.framework.batchprocess.entity.type |
| Notificacion | pe.gob.mpfm.notification.entity.type |
| Eventos | pe.gob.mpfm.httpevent.publish.entity.type |

6.5 Beans JSF

Deben ser nombrados con un nombre que represente la funcionalidad a realizar y deben terminar en el sufijo Bean.

| Modulo | Ejemplo |
|----------------------|--------------------------|
| Feriados | HolidayMgmtBean |
| Procesos programados | SchedulerProcessMgmtBean |

6.6 Beans Boundary

Deben ser nombrados con un nombre de represente al módulo de negocio que permiten administrar y deben terminar en el sufijo Boundary

| Modulo | Ejemplo |
|----------------------|---------------------------|
| Reportes | ReportMgmtBoundary |
| Procesos de negocio | BusinessProcessesBoundary |
| Procesos programados | SchedulerBoundary |

6.7 Beans Controller

Deben ser nombrados con un nombre que representa la funcionalidad a realizar y deben terminar en el sufijo Controller

| Modulo | Ejemplo |
|----------------------|-------------------------------|
| Reportes | ReportManageController |
| Procesos de negocio | MonitoringProcessesController |
| Procesos programados | SchedulerController |

6.8 Archivos Properties

Los archivos properties en cada proyecto web deberán ser llamados Mensajes_es y Mensajes_en. Estos archivos deberán estar dentro de la carpeta resource del paquete java para que pueda estar empaquetados por maven y disponible en el classpath de la aplicación al momento de deployar.

7. Componentes de la Arquitectura

7.1 Capa de Presentación

Se detallara las convenciones de trabajo para los frameworks utilizados ya sea JSF/Primefaces o Angular 2.x

7.1.1 JSF

Por default el nombre de los beans que podrán ser usados en el EL en las páginas xhtml cuando usan las anotaciones son el de las clases con la primera letra en minúscula

Conversation: Usado para beans que tienen que mantener los datos más de una petición y entre distintas páginas. Se usa la anotación `@ConversationBean`.

```
@ConversationBean
public class SchedulerProcessMgmtBean extends BaseBean implements Serializable{
```

View: Usado para beans que tienen que mantener los datos en más de una petición pero en la misma página. Se usa la anotación `@ViewBean`.

```
@ViewBean
public class ReporteDynamicFilterBean extends BaseBean implements Serializable {
```

7.2 Seguridad de la Capa de Negocio

Los servicios expuestos en la capa Boundary(RestFul) deberán ser validados utilizando las anotaciones `@TokenAuthenticated` y `@UserAudit`

```
@Path("aplicaciones")
@Stateless
public class AplicacionResource {

    /** La crud service. */
    @EJB
    CrudServiceController crudService;

    /**
     * Crear aplicacion.
     *
     * @param aplicacion el aplicacion
     * @return the response
     */
    @POST
    @Consumes(MediaType.APPLICATION_JSON)
    @Produces(MediaType.APPLICATION_JSON)
    @TokenAuthenticated
    @UserAudit
```

```
public Response crearAplicacion(Aplicacion aplicacion) {  
    aplicacion = crudService.create(aplicacion);  
    return Response.ok(aplicacion.getIdAplicacionPk()).build();  
}
```

@TokenAuthenticated: Esta anotación hace que cada método que la está utilizando sea validado con el token de seguridad que previamente debió ser obtenido del sistema de seguridad implementado.

@UserAudit: Obtiene el nombre de usuario e ip extraídos del Token de Seguridad y esa información es agregada al objeto UserTrack el cual se encargara de colocar los datos de auditoria de registros en las entidades al momento de realizar una operación de persist o merge.

7.3 Utilización del CrudServiceController

El Controller CrudServiceController que es un EJB implementa métodos comunes para la realización de operaciones con la base de datos.

```
@Stateless  
public class CrudServiceController {  
  
    /** La em. */  
    @Inject @DbApplication  
    protected EntityManager em;  
  
    /** La transaction registry. */  
    @Resource  
    private TransactionSynchronizationRegistry transactionRegistry;  
  
    /**  
     * Creates the.  
     *  
     * @param <T> el tipo generico  
     * @param t el t  
     * @return the t  
     */  
    public <T> T create(T t) {  
        Object object =  
transactionRegistry.getResource(RegistryContextHolder.USER_TRACK);  
        return create(t, object);  
    }  
}
```


7.3.1 Configuración

Para su utilización se deberá configurar el nombre de la unidad de persistencia de la siguiente manera

```
public class ProducerDataBase {

    /** La em. */
    @Produces
    @DbApplication
    @PersistenceContext(unitName = "osisinventario")
    private EntityManager em;

    /**
     * Instancia un nuevo producer data base.
     */
    public ProducerDataBase() {
        // TODO Auto-generated constructor stub
    }

}
```

7.3.2 Utilización

Para su utilización se deberá inyectar el componente en el Boundary o Controller que lo necesite por ejemplo:

```
@Path("parametros")
@Stateless
public class ParametroResource {

    /** La crud service. */
    @EJB
    CrudServiceController crudService;

    /**
     * Obtener parametros de maestro.
     *
     * @param idMaestro el id maestro
     * @return the json array
     */
    @GET
    @Path("maestro/{id}")
    @Produces(MediaType.APPLICATION_JSON)
    public JSONArray obtenerParametrosDeMaestro(@PathParam("id") Integer idMaestro){
        Map<String, Object> parametros = new HashMap<>();
        parametros.put("master", idMaestro);
        List<TablaParametro> listaParametros =
        crudService.findWithNamedQuery(TablaParametro.LISTAR_PARAMETROS, parametros);
        JSONArrayBuilder array = Json.createArrayBuilder();
        for(TablaParametro parametroItem : listaParametros){
            JsonObject parametro = Json.createObjectBuilder()
                .add("idParametro", parametroItem.getIdTablaParametroPk())
                .add("nombre", parametroItem.getNombre()).build();
            array.add(parametro);
        }
        return array.build();
    }
}
```

Utilizando el método `findWithNamedQuery` el cual recibe como parámetro el nombre de la query ya definida en la entidad.

```
@Entity
@Table(name="tabla_parametro")
@NamedQuery(name="TablaParametro.buscarParametros", query="SELECT t FROM TablaParametro t where t.tablaMaestra.idTablaMaestraPk=:master")
public class TablaParametro implements Serializable {

    @GET
    @Path("/{id}")
    @Consumes(MediaType.APPLICATION_JSON)
    @Produces(MediaType.APPLICATION_JSON)
    public Response obtenerAplicacion(@PathParam("id") Long idAplicacion) {
        return Response.ok(crudService.find(Aplicacion.class, idAplicacion)).build();
    }
}
```

Utilizamos el método `find` para devolver toda la entidad.

7.3.3 Auditoria

Cuando utilizamos las operaciones para registrar o modificar una entidad utilizando los servicios del `CrudServiceController` automáticamente serán colocados los valores en los campos de auditoria de cada entidad

Boundary

```
@POST
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
@TokenAuthenticated
@UserAudit
public Response crearAplicacion(Aplicacion aplicacion) {
    aplicacion = crudService.create(aplicacion);
    return Response.ok(aplicacion.getIdAplicacionPk()).build();
}
```

Controller

```
@ApplicationScoped
public class AplicacionController {

    @EJB
    CrudServiceController crudService;

    public Aplicacion modificarAplicacion(Long idApp, Aplicacion aplicacion) {
        Aplicacion appOld = crudService.find(Aplicacion.class, idApp);
        appOld.setNombre(aplicacion.getNombre());
        appOld.setTecnologia(aplicacion.getTecnologia());
        appOld.setUrlSvn(aplicacion.getUrlSvn());
        appOld.setVersion(aplicacion.getVersion());
        return crudService.update(appOld);
    }
}
```

Campos de auditoria que son persistidos

```
/** usuario modificacion. */
@Column(name="usuario_modificacion")
private String usuarioModificacion;

/** La fecha modificacion. */
@Column(name="fecha_modificacion")
private LocalDateTime fechaModificacion;

/** La ip modificacion. */
@Column(name="ip_modificacion")
private String ipModificacion;

/** La menu modificacion. */
@Column(name="menu_modificacion")
private String menuModificacion;
```

7.4 Mapeo utilizando JPA

Para el mapeo de las entidades utilizamos JPA 2.1 como framework y utilizamos el proveedor del servidor de aplicaciones que es Hibernate, pero como trabajamos solo con JPA debería ser sencillo migrar a otros servidores que utilicen otros implementadores como EclipseLink, TopLink, etc.

7.4.1 Auditoria

Implementando la interface Auditable permitirá configurar como cada entidad deberá obtener los datos de auditoria al momento de ser persistida o actualizada.

```
/**
 * <ul>
 * <li>Copyright 2016 Ministerio Publico - Fiscalia de la Nacion. Toc
 * </li>
 * </ul>
 *
 * La Interface Auditable.
 *
 * @author OSIS
 * @version 1.0 , 08/04/2016
 */
public interface Auditable {

    /**
     * Establece el audit.
     *
     * @param loggerUser el new audit
     */
    public void setAudit(UserTrack loggerUser);

    /**
     * Gets the list for audit.
     *
     * @return the list for audit
     */
    public Map<String,List<? extends Auditable>> getListForAudit();
}
```

Ilustración 9 Interface a implementar

Cada entidad deberá implementar el método a continuación

```
/* (non-Javadoc)
 * @see
 * pe.gob.mpfن.entitymanager.auditoria.Auditable#setAudit(pe.gob.mpfن.entitymanager.auditoria.entity.User
 * Track)
 */
@Override
public void setAudit(UserTrack userTrack) {
    this.fechaModificacion=userTrack.getAuditTime();
    this.usuarioModificacion=userTrack.getUserName();
    this.ipModificacion=userTrack.getIpAddress();
    this.menuModificacion=userTrack.getMenuOption();
}
```

Si se tienen mapeados listados de objetos con cascade=CascadeType.ALL que no es recomendable se debería implementar el método de la siguiente forma por ejemplo:

```
@Override
public Map<String, List<? extends Auditable>> getListForAudit() {

    HashMap<String,List<? extends Auditable>> detailsMap = new HashMap<String, List<? extends Auditable>>();
    detailsMap.put("holderFile",holderFiles);
    detailsMap.put("pepPerson", pepPersons);
    detailsMap.put("representedEntity", representedEntities);
    return detailsMap;
}
```

7.4.2 Concurrencia

Las entidades deberán agregar un campo de tipo Long que se llamara por defecto versión y estará mapeado con la siguiente anotación

```
/** La version. */
@Version
private Long version;
```

Esto permitirá que alguna operación de modificación ejecutada con el controller CrudServiceController.update dispare una excepción al intentar modificar la entidad con datos antiguos(es decir ya fue actualizado y tiene otro número de versión en la base de datos)

Para aquellos que usen modificaciones a través de JPQL o ejecución de queries nativas JPA no puede controlar la concurrencia de updates porque JPQL es ejecutado en un contexto diferente al que usamos cuando modificamos algo con merge en el EntityManager, en ese caso en su query de update tendrán que agregar el “entity.version=entitiy.version + 1” para mantener sincronizado el conteo de actualizaciones.

7.4.3 Tipos de datos LocalDateTime y LocalDate

A partir de Java 8 se agregó un nuevo api para el manejo de fechas basado en el framework Joda-Time que permite hacer operaciones con fechas de una manera mas amigable y productiva.

Es por eso que el campo de auditoria “fechaModificacion” es del tipo LocalDateTime que permite guarda fecha, hora, minutos, segundo; si solo se desea guarda fecha se debería usar el tipo LocalDate.

Por defecto JPA no reconoce los nuevos tipos LocalDateTime ni LocalDate como fechas para el mapeo, pero el componente mpfn-entitymanager ya implementa un converter para realizar la conversión automática de estos tipos al tipo Date de manera transparente para nosotros por eso para estos nuevos tipos de datos de fecha no se debe colocar la anotación `@Temporal(TemporalType.TIMESTAMP)`

7.4.4 Bean Validation

La validación de datos de entrada recibidas del usuario para mantener la integridad de los datos es un parte importante de la lógica de la aplicación. Validación de datos puede llevarse a cabo en las distintas capas de la aplicación desde la presentación hasta la capa de negocios.

JavaBean Validation (Bean Validation) es el nuevo modelo de validación disponible como parte de la plataforma Java EE 7. El modelo de Bean Validation es soportado por restricciones en forma de anotaciones aplicadas a campos, métodos o clases de un JavaBean componente.

Se pueden crear nuevas restricciones definidas por el usuario basadas en las restricciones definidas en la especificación.

| Constraint | Descripción | Ejemplo |
|-------------|---|--|
| @DecimalMax | El valor del campo o propiedad debe ser un valor decimal menor o igual al número en el valor de la anotación | @DecimalMax("30.00") BigDecimal discount |
| @DecimalMin | El valor del campo o propiedad debe ser un valor decimal mayor o igual al número en el valor de la anotación | @DecimalMin("5.00") BigDecimal discount |
| @Digits | El valor del campo o propiedad debe ser un número con un especificado rango, el elemento integer especifica el máximo de dígitos enteros para el numero, el elemento fraction especifica el máximo de dígitos | @Digits(integer=6, fraction=2) BigDecimal price |

| | | |
|----------|--|--|
| | decimales para el numero | |
| @Future | El valor del campo o propiedad debe ser una fecha futura | @Future Date eventDate |
| @Max | El valor del campo o propiedad debe ser un valor entero menor o igual al número en el valor de la anotación | @Max(10) int quantity |
| @Min | El valor del campo o propiedad debe ser un valor entero mayor o igual al número en el valor de la anotación | @Min(5) int quantity |
| @NotNull | El valor del campo o propiedad debe ser diferente a nulo | @NotNull String username |
| @Null | El valor del campo o propiedad debe ser nulo | @Null String unusedString |
| @Past | El valor del campo o propiedad debe ser una fecha pasada | @Past Date birthday |
| @Pattern | El valor del campo o propiedad debe coincidir con la expresión regular definida en la anotación | @Pattern(regexp="(\\d{3}\\d{3}-\\d{4})") String phoneNumber |
| @Size | El valor del campo o propiedad debe coincidir con el tamaño especificado según los límites. Si el campo o propiedad es un String el tamaño del string es evaluado, si es una colección evalúa el tamaño de la colección, si es un Map el tamaño del Map es evaluado. | @Size(min=2, max=240) String briefMessage |

Ilustración 10 Tabla con la lista de anotaciones para validaciones

7.5 Logging

Utilizamos la clase MpfnLogger que está configurada para abstraernos de la implementación real de la librería de loggins (logj4, slf4j,etc)

Para usarlo en un EJB o un bean CDI simplemente usar:

```
@Inject  
MpfnLogger log;
```

Para utilizarlo en bean JSF se deberá invocarlo asi:

```
@Inject  
private transient MpfnLogger log;
```

Esto porque como es un objeto que serializa su estado no debe serializar una dependencia de tipo logger;

Se debe tratar de no utilizar System.out.printXXXX para pintar logs.

7.6 Gestión de Errores Capa Negocio

Solo se deberán capturar excepciones de negocio es decir excepciones disparadas por casuísticas propias de reglas e implementadas como una clase que hereda de Exception por ejemplo:

```
@ApplicationException(rollback=true)  
public class ServiceException extends Exception {
```

Esas excepciones son las únicas que se deben manejar manualmente ya que el componente mpfn-core implementa un mapeador de excepciones runtime las cuales son capturadas, loggeadas y devueltas al llamador del servicio con códigos de error http.

```
@Provider
public class EJBExceptionHandler implements ExceptionMapper<EJBException> {

    @Inject
    MpfnLogger log;

    /* (non-Javadoc)
     * @see javax.ws.rs.ext.ExceptionMapper#toResponse(java.lang.Throwable)
     */
    @Override
    public Response toResponse(EJBException ex) {
        Throwable cause = ex.getCause();
        if (cause instanceof OptimisticLockException) {
            OptimisticLockException actual = (OptimisticLockException) cause;
            log.error(actual.getMessage() + actual.getEntity() != null ? actual.getEntity().toString() : "");
            return Response.status(Response.Status.CONFLICT)
                .header(HeaderValuesType.CAUSE.getValue(), "El registro ya fue procesado : ")
                .build();
        } else if (cause instanceof EntityNotFoundException) {
            EntityNotFoundException actual = (EntityNotFoundException) cause;
            log.error(actual.getMessage());
            return Response.status(Response.Status.NOT_FOUND)
                .header(HeaderValuesType.CAUSE.getValue(), "Entidad no se encuentra registrada ").build();
        } else {
            //Otra exception EJB
            log.error(cause.getMessage());
            log.error(ex.toString());
            return Response.serverError();
        }
    }
}
```

Ilustración 11 Mapeador de excepciones RestFul

7.7 Configuración de Parámetros de Despliegue

La aplicación utiliza la inyección de dependencia proporcionada por CDI para poder configurar parámetros de despliegue es decir que solo deberían modificar por cambios en el despliegue de la aplicación.

Para estas configuraciones se utiliza el archivo **ApplicationsConfigurations.properties** y para referenciar los valores en el código se utiliza la anotación **@Configurable**

```
1 #####Configuration of Application#####
2 serverLdapHost=172.16.104.112
3 serverLdapPort=389
4 rootUserLdap=cn=manager,ou=Internal,dc=mpfn,dc=local
5 rootPasswordLdap=anhsVBtLrMoD4/ZD
6 usersGroupDNLdap=ou=Users,ou=Accounts,dc=mpfn,dc=local
7 userDNLdap=cn={0},ou=Users,ou=Accounts,dc=mpfn,dc=local
8 ##### Pool Ldap connections #####
9 rootPoolMaxConnections=20
10 appPoolMaxConnections=20
11 ##### Implementation Ldap #####
12 ##### valores disponibles OpenLdap y ActiveDirectory
13 ldapBeanImplementation=OpenLdap
```

Ilustración 12 Ejemplo de definición de parámetros


```
public abstract class ConfigurableProvider {  
  
    @Inject @Configurable  
    protected String serverLdapHost;  
  
    @Inject @Configurable  
    protected Integer serverLdapPort;  
  
    @Inject @Configurable  
    protected String rootUserLdap;  
  
    @Inject @Configurable  
    protected String rootPasswordLdap;  
  
    @Inject @Configurable  
    protected String usersGroupDNLdap;  
  
    @Inject @Configurable  
    protected String userDNLdap;  
  
    @Inject @Configurable  
    protected Integer rootPoolMaxConnections;  
  
    @Inject @Configurable  
    protected Integer appPoolMaxConnections;  
}
```

Ilustración 13 Ejemplo de utilización en el código

Como convención el nombre del key declarado en el archivo properties será el mismo nombre de la variable que será declarada en la clase que desea usar el parámetro por ejemplo el key serverLdapHost es referenciado en la clase como

```
@Inject @Configurable  
String serverLdapHost;
```

Es posible referenciar a un parámetro definido en el archivo properties con un nombre de variable diferente por ejemplo

```
@Inject @Configurable("app.web.audit.version")  
String aplicacionVersion;
```

8. Configuraciones de Herramientas para Desarrollo

A continuación se describe la configuración de algunas herramientas que son necesarias para el correcto desarrollo y utilización de los componentes de la arquitectura.

8.1 JDK 8

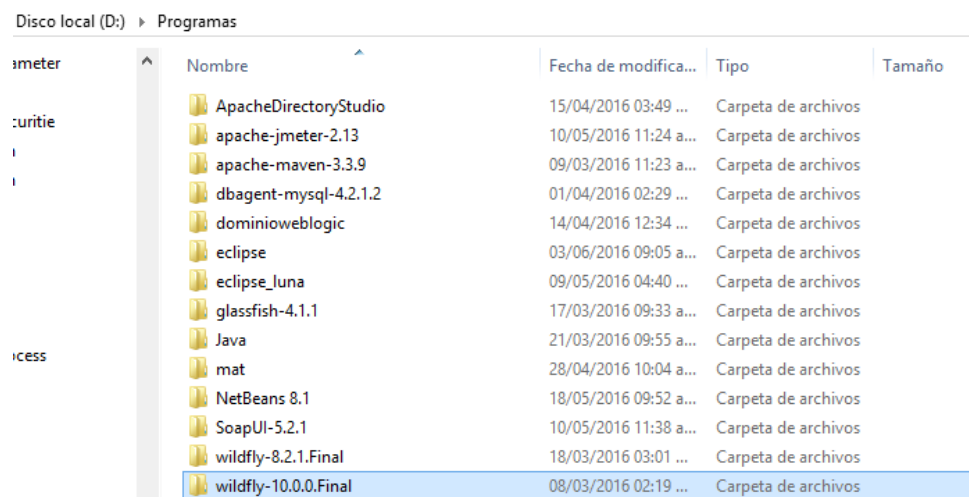
Como mínimo se trabajara con la versión jdk-8u74 de 64 bits

8.2 Wildfly 10

Para el desarrollo de las aplicaciones trabajaremos con Wildfly 10 en modo standalone que requiere el JDK 8 para poder trabajar y se recomienda tener la variable JAVA_HOME apuntando al lugar donde está instalado el JDK.

8.2.1 Instalación

- Descomprimir el zip que contiene al servidor en una carpeta que no tenga nombres con espacios por ejemplo



| Disco local (D:) > Programas | | | | |
|------------------------------|-----------------------|-----------------------|---------------------|--------|
| | Nombre | Fecha de modifica... | Tipo | Tamaño |
| | ApacheDirectoryStudio | 15/04/2016 03:49 ... | Carpeta de archivos | |
| | apache-jmeter-2.13 | 10/05/2016 11:24 a... | Carpeta de archivos | |
| | apache-maven-3.3.9 | 09/03/2016 11:23 a... | Carpeta de archivos | |
| | dbagent-mysql-4.2.1.2 | 01/04/2016 02:29 ... | Carpeta de archivos | |
| | dominioweblogic | 14/04/2016 12:34 ... | Carpeta de archivos | |
| | eclipse | 03/06/2016 09:05 a... | Carpeta de archivos | |
| | eclipse_luna | 09/05/2016 04:40 ... | Carpeta de archivos | |
| | glassfish-4.1.1 | 17/03/2016 09:33 a... | Carpeta de archivos | |
| | Java | 21/03/2016 09:55 a... | Carpeta de archivos | |
| | mat | 28/04/2016 10:04 a... | Carpeta de archivos | |
| | NetBeans 8.1 | 18/05/2016 09:52 a... | Carpeta de archivos | |
| | SoapUI-5.2.1 | 10/05/2016 11:38 a... | Carpeta de archivos | |
| | wildfly-8.2.1.Final | 18/03/2016 03:01 ... | Carpeta de archivos | |
| | wildfly-10.0.0.Final | 08/03/2016 02:19 ... | Carpeta de archivos | |

Ilustración 14 Descomprimir en carpeta sin espacios en blanco en los nombres

- Instalar un usuario administrador para gestionar el wildfly local a través de la consola, para eso ejecutamos el archivo add-user.bat que se encuentra dentro de la carpeta bin y crearemos un usuario del tipo “a”

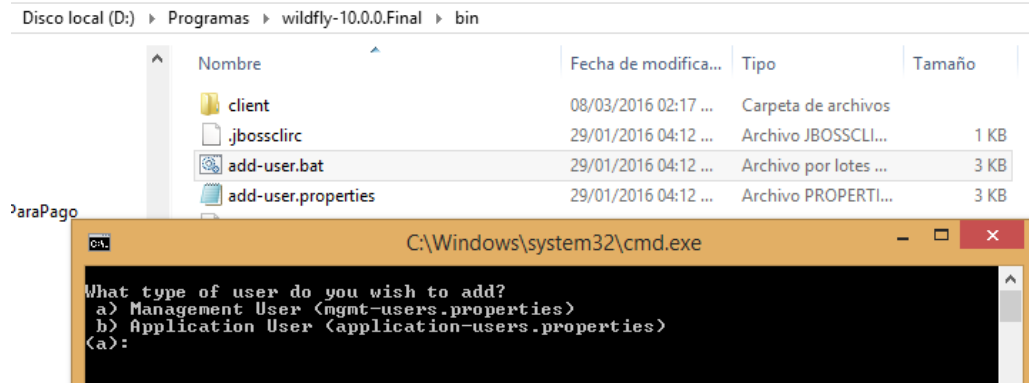


Ilustración 15 Creación usuario administrador

- Iniciamos el wildfly ejecutando el archivo standalone.bat que se encuentra dentro de la carpeta bin

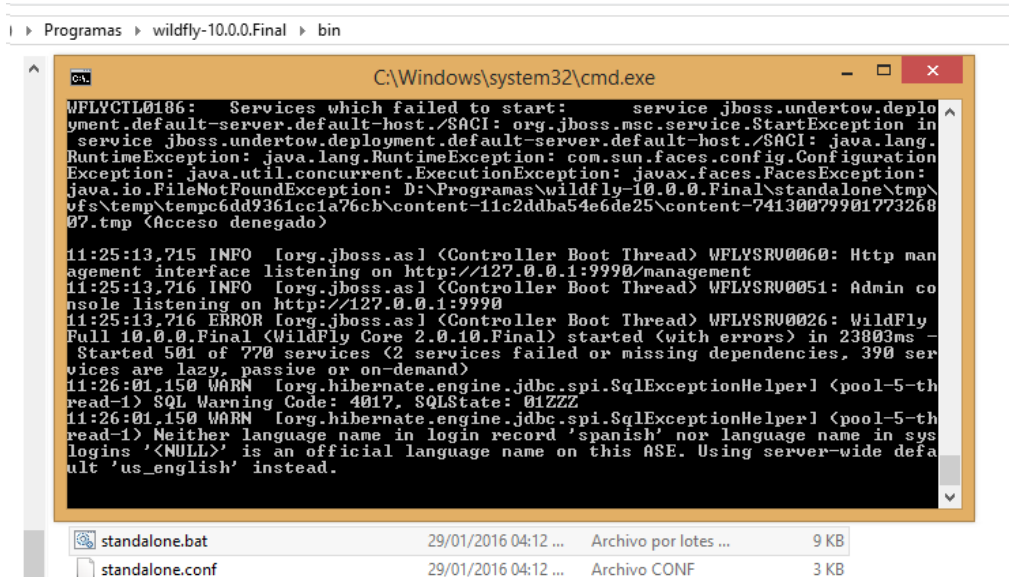


Ilustración 16 Iniciando por consola

- Colocamos la url localhost:9990/console para ingresar a la parte administrativa del wildfly

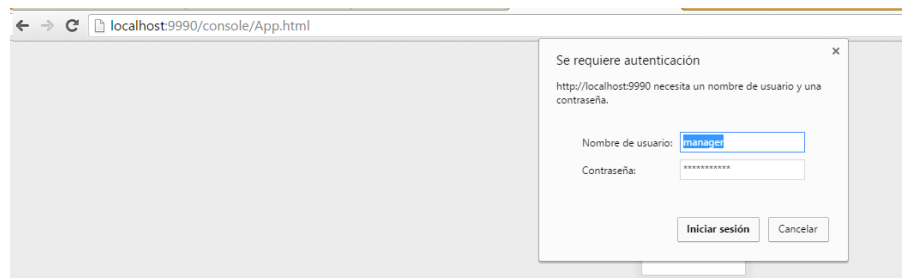


Ilustración 17 Ingresar usuario y password creados

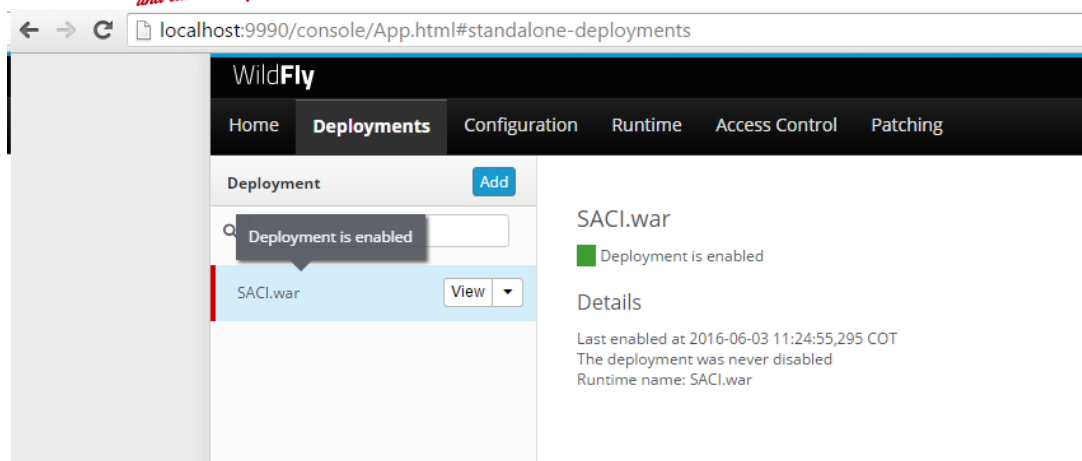


Ilustración 18 Consola administrativa

8.2.2 Configuración Datasource

- Instalar el driver sybase como módulo, para eso copiar el contenido de la carpeta com que se encuentra dentro de la carpeta driver sybase a la carpeta modules del wildfly

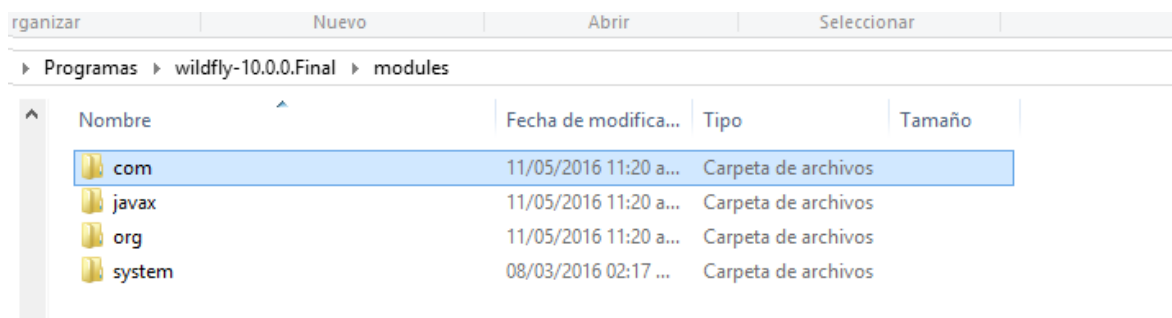


Ilustración 19 Copiar a la carpeta modules del wildfly

- Modificar el archivo standalone.xml dentro de la carpeta standalone/configuration del wildfly agregando dentro del tag <datasources><drivers> lo siguiente:

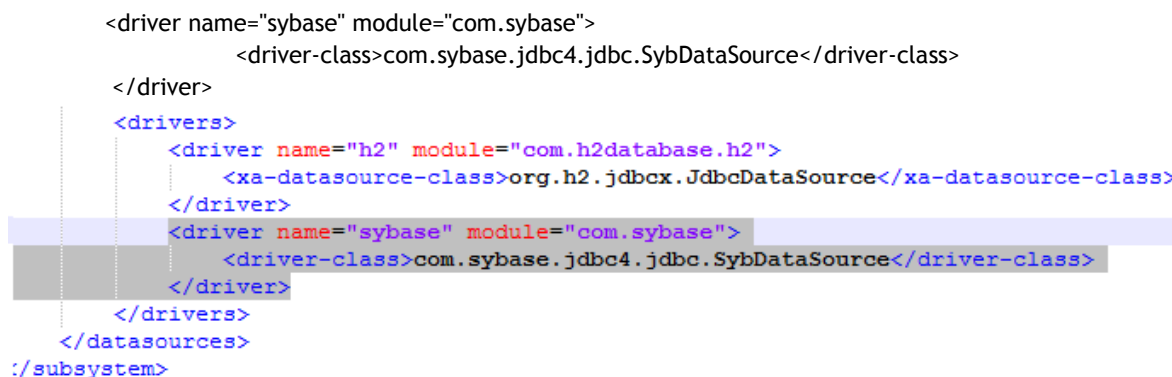


Ilustración 20 Agregar el driver a la configuración

- Al momento de crear un datasource asegurarse que el atributo jta sea igual a true y que el atributo transaction isolation sea igual a **TRANSACTION_READ_COMMITTED**

The screenshot shows the 'DATASOURCES' configuration window in Eclipse. The title bar says 'JDBC datasource 'jdbc/saci' (enabled)'. Below the title, it says 'JDBC datasource configurations.' and there is a 'Disable' button. A tabbed interface is shown with 'Attributes', 'Connection', 'Pool', 'Security', 'Properties', 'Validation', 'Timeouts', and 'Statements'. The 'Connection' tab is selected. In the top right of the tab, there is a 'Test Connection' button and a 'Need Help?' link. On the left, there is an 'Edit' icon. The configuration fields are as follows:

| | |
|------------------------|---------------------------------------|
| Connection URL: | jdbc:sybase:Tds:172.16.0.28:5000/saci |
| New Connection Sql: | |
| Transaction Isolation: | TRANSACTION_READ_COMMITTED |
| Use JTA?: | true |

Ilustración 21 Atributos de datasources

8.3 Eclipse Mars

Los proyectos estructurados con Maven son agnósticos a los IDEs de desarrollo (Eclipse, Netbeans, IntelliJ IDEA) pero para estandarizar y dar soporte al desarrollo de los proyectos se utilizara Eclipse Mars para la implementación de los aplicativos.

Para poder comenzar a trabajar es necesario realizar algunas configuraciones y la instalación de plugins necesarios para el trabajo.

8.3.1 Configuración Proxy

Ingresa a la opción Windows-> Preferences -> General -> Network Connections luego cambiar el valor del combo Active Provider a Manual y configurar el proxy para conexión a internet desde el eclipse

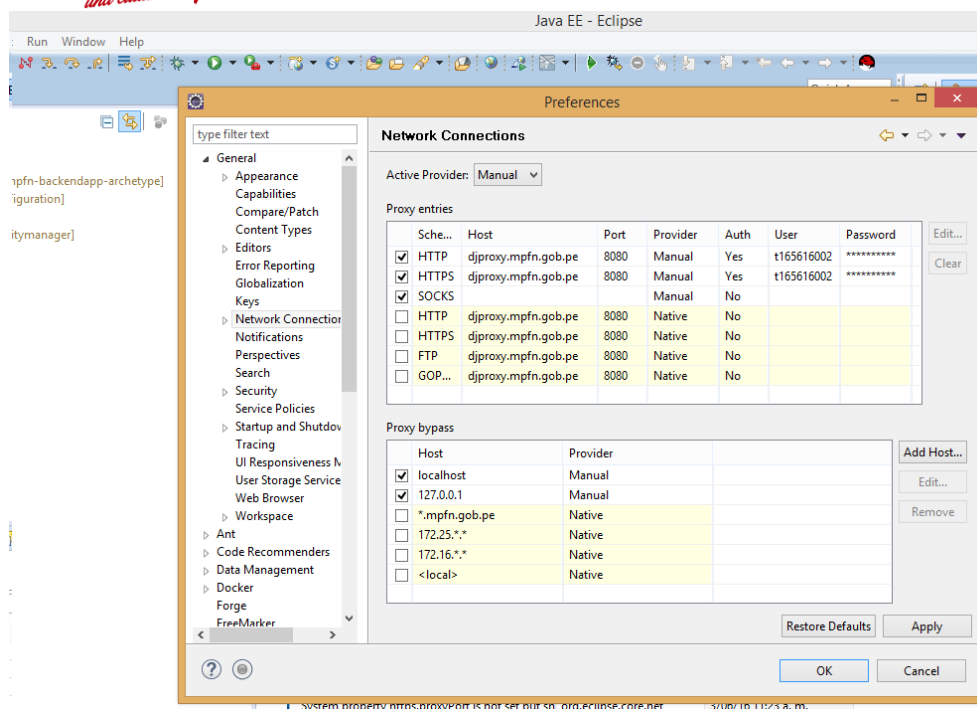


Ilustración 22 Configurar proxy

8.3.2 Configuración JDK

Ingresa a la opción Windows-> Preferences -> General ->Java -> Installed JREs y agregar la ruta de su JDK no JRE que instalaron dando click al botón Add, una vez agregado seleccionar el check para que queda como la instalación por defecto.

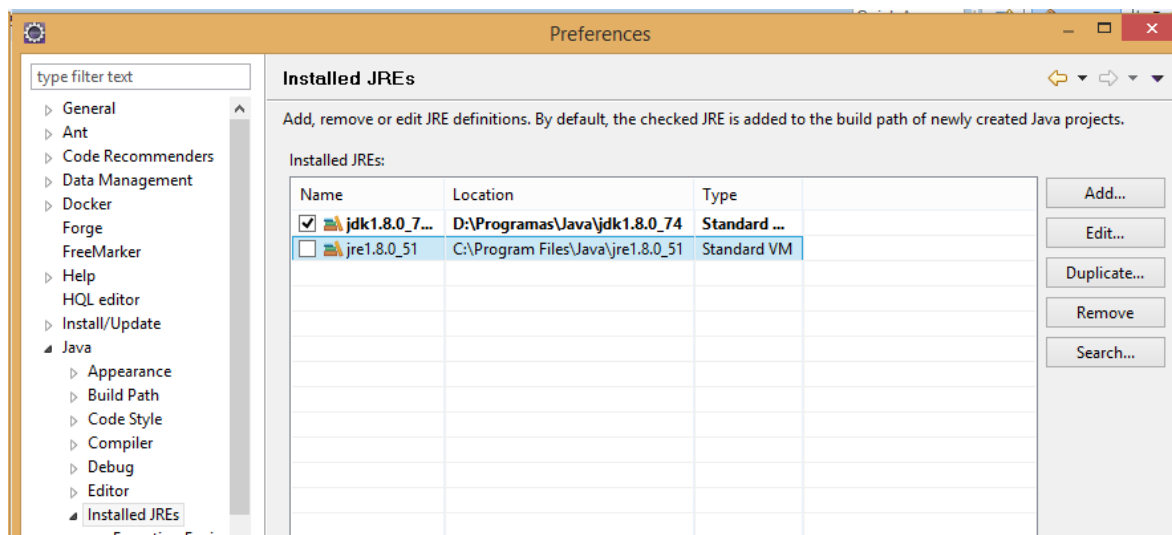


Ilustración 23 Configuración utilización del JDK

8.3.3 Jboss Tools

- Ingresar a la opción Help -> Eclipse Marketplace

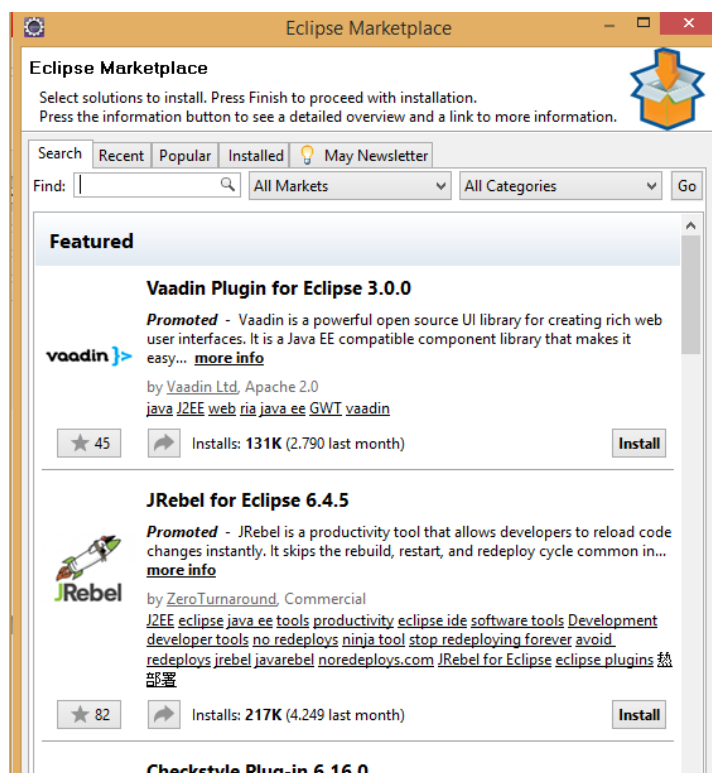


Ilustración 24 Eclipse Marketplace

- Buscar jboss tools e instalar la última versión disponible

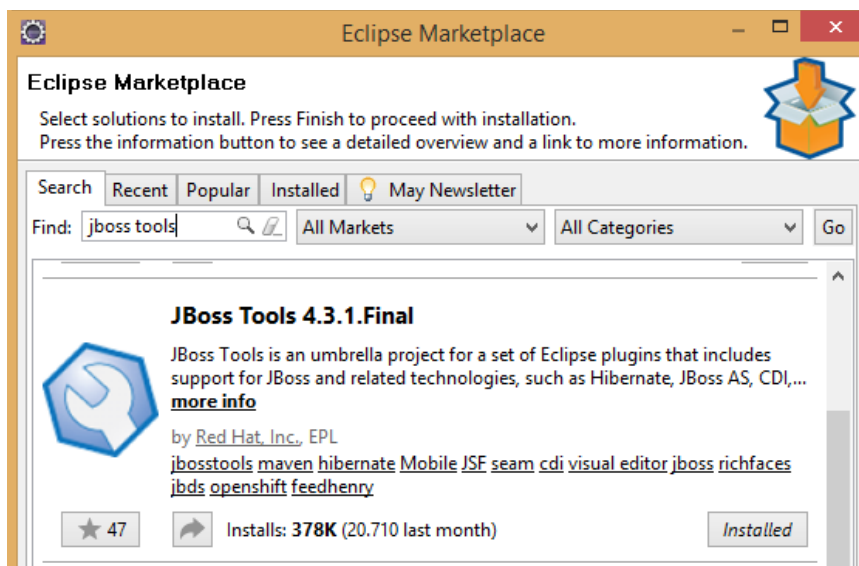


Ilustración 25 Instalar Jboss tools

8.3.4 Subclipse

- Ingresar a la opción Help -> Eclipse Marketplace buscar subclipse y proceder a instalar la última versión.

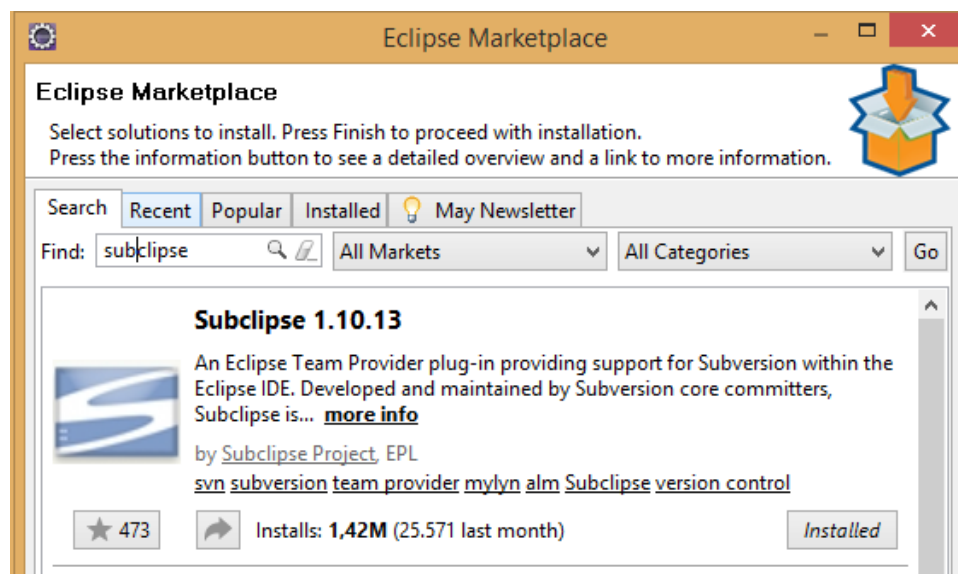


Ilustración 26 Instalar Subclipse

8.3.5 Jautodoc

- Ingresar a la opción Help -> Eclipse Marketplace buscar jautodoc y proceder a instalar la última versión.

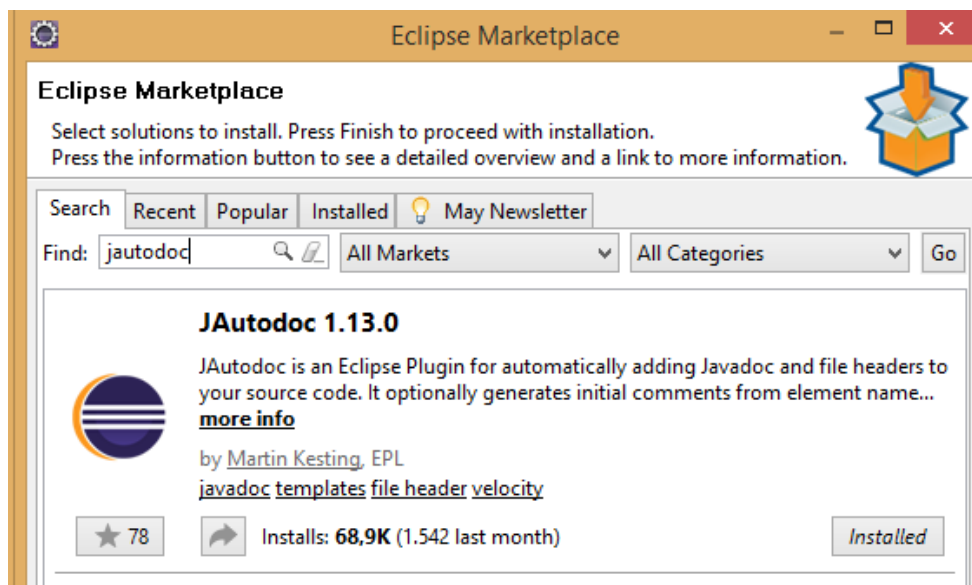


Ilustración 27 Instalar Jautodoc

8.3.6 Configurar Maven settings.xml

Ingresa a la opción Windows-> Preferences -> Maven -> User Settings y configurar la ruta donde estará el archivo settings.xml el cual tiene las configuraciones para usar el repositorio local Nexus para administrar las dependencias Maven.

Por lo general la ruta donde se debe crear el archivo es C:\Users\Username\.m2

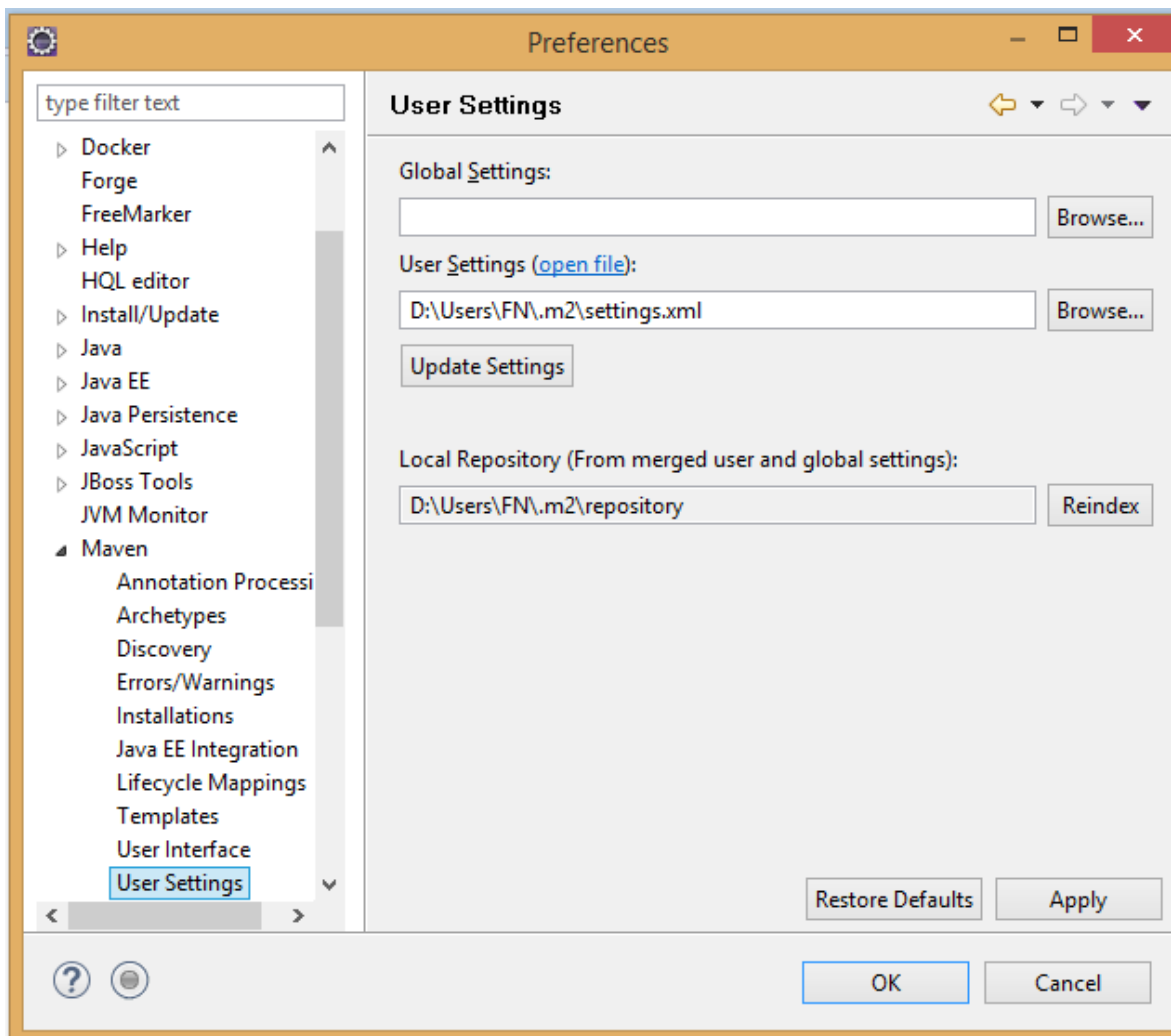


Ilustración 28 Configurar archivo settings

A continuación se muestra el contenido del archivo settings.xml para poder utilizar el servidor Nexus como repositorio Maven y resolver desde ahí las dependencias a componentes de terceros así como a componentes propios creados.

Asegurarse de configurar su usuario y password del proxy para el internet en el archivo

```
<settings>
  <proxies>
    <proxy>
      <id>example-proxy</id>
      <active>true</active>
      <protocol>http</protocol>
      <host>djproxy.mpfh.gob.pe</host>
      <port>8080</port>
      <username>usuarioproxy</username>
      <password>claveproxy</password>
      <nonProxyHosts>www.google.com|*.example.com</nonProxyHosts>
    </proxy>
  </proxies>
  <mirrors>
    <mirror>
      <!--This sends everything else to /public -->
      <id>nexus</id>
      <mirrorOf>*</mirrorOf>
      <url>http://172.25.44.83:8081/repository/maven-public/</url>
    </mirror>
  </mirrors>
  <profiles>
    <profile>
      <id>nexus</id>
      <!--Enable snapshots for the built in central repo to direct -->
      <!--all requests to nexus via the mirror -->
      <repositories>
        <repository>
          <id>central</id>
          <url>http://central</url>
          <releases><enabled>true</enabled></releases>
          <snapshots><enabled>true</enabled></snapshots>
        </repository>
      </repositories>
      <pluginRepositories>
        <pluginRepository>
          <id>central</id>
          <url>http://central</url>
          <releases><enabled>true</enabled></releases>
          <snapshots><enabled>true</enabled></snapshots>
        </pluginRepository>
      </pluginRepositories>
    </profile>
  </profiles>
  <activeProfiles>
    <!--make the profile active all the time -->
    <activeProfile>nexus</activeProfile>
  </activeProfiles>
  <!-- Configuración para deploys maven -->
</settings>
```

8.3.7 Configuración de Repositorio de Archetypes

Ingresa a la opción Windows-> Preferences -> Maven -> Archetypes y dar click al botón Add Remote Catalog y en el campo Catalog File agregar

<http://172.25.44.83:8081/repository/maven-public/>

En el campo Description agregar mpfn-archetype y dar click al botón OK

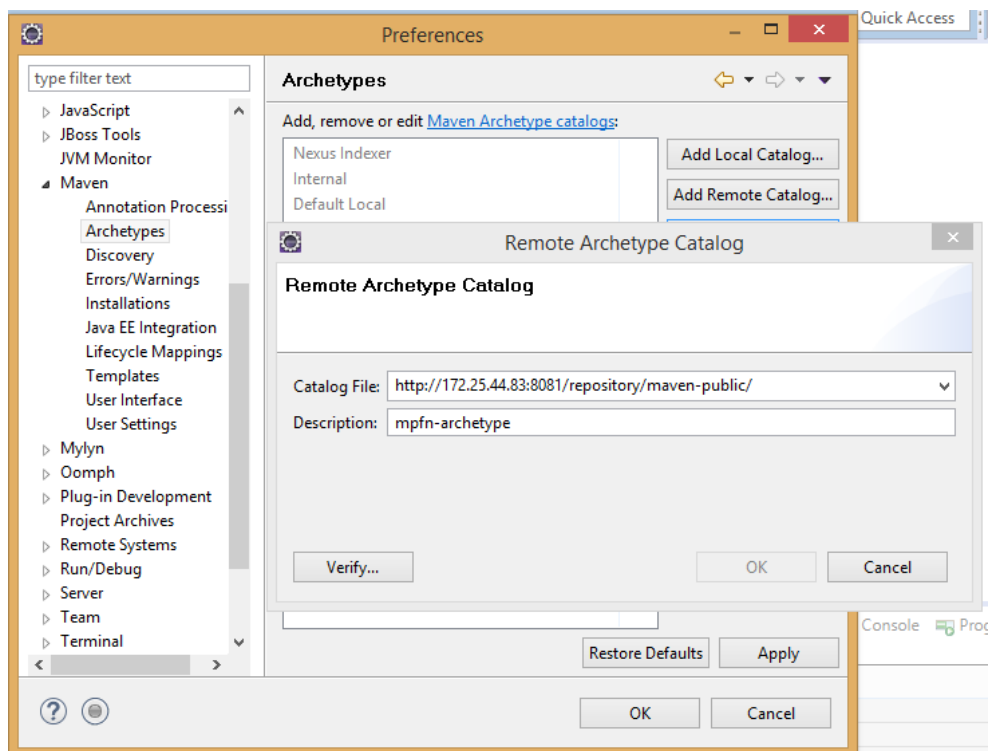


Ilustración 29 Agregar repositorio archetype

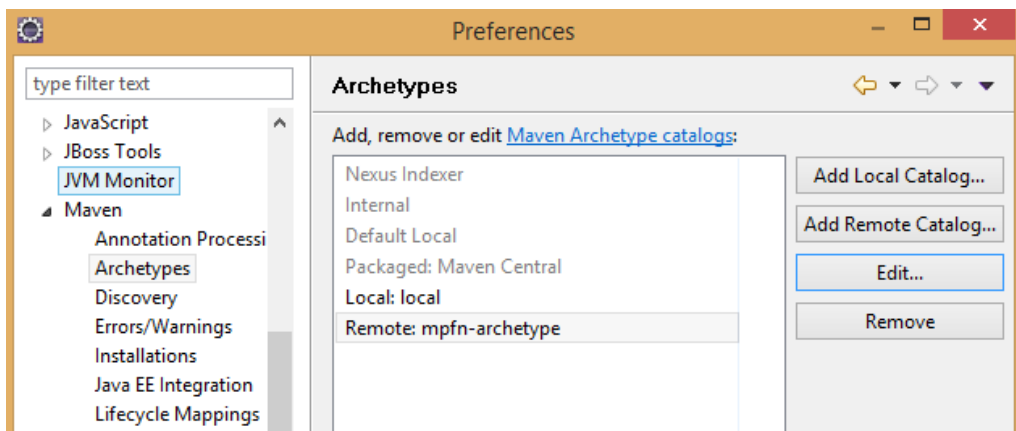


Ilustración 30 Repositorio agregado

8.3.8 Configuración del Jautodoc

Ingresa a la opción Windows-> Preferences -> Java -> Jautodoc -> Templates , darle click al botón Import y seleccionar el archivo jautodoc_templatesSpanish.xml que tiene configurado la cabecera de documentación de las clases

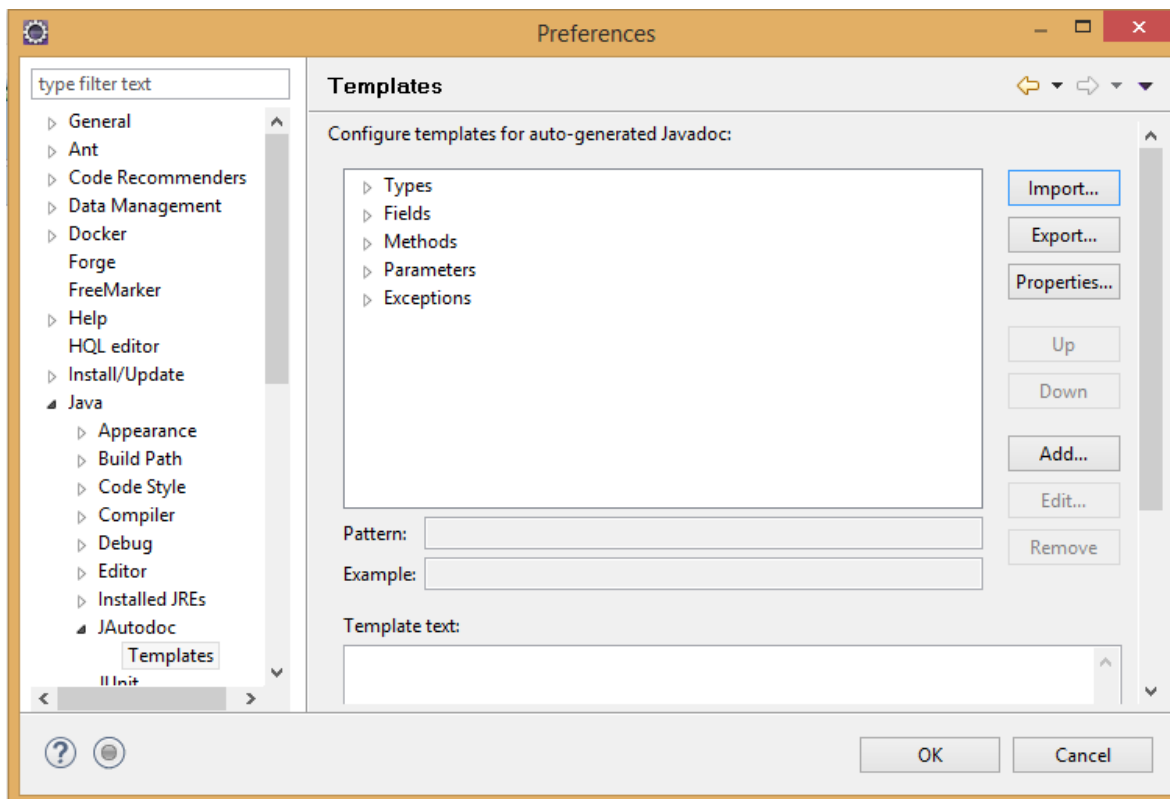


Ilustración 31 Configurar Plantilla

9. Pasos para el Desarrollo

9.1 Acceso al Código de los Proyectos

Para trabajar con proyectos existentes es necesario obtener el código fuente desde el servidor SVN del proyecto que se desea, para eso se necesita agregar el repositorio svn al eclipse si todavía no se tiene y después obtener la última versión del código fuente.

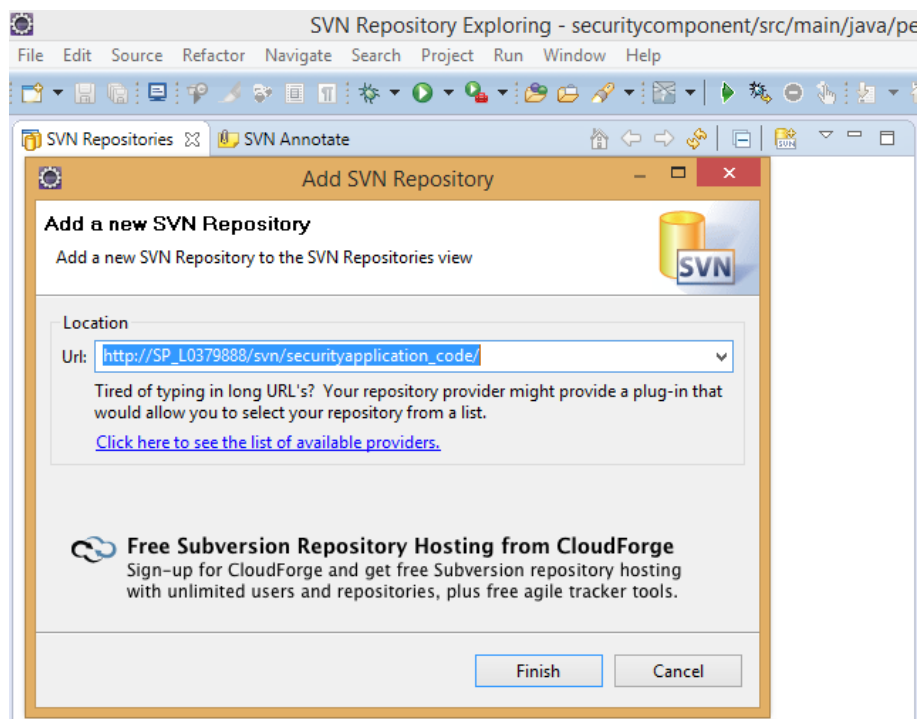
9.1.1 Agregar Repositorio SVN

Cambiar a la perspectiva SVN Repository Exploring y dar click al botón svn+



Ilustración 32 Agregar Repositorio

En el campo URL agregar la ruta del repositorio al cual se quiere tener acceso



Las credenciales que se pedirán serán las mismas que deben habérselas generado para tener acceso a los repositorios SVN

9.1.2 Checkout del Código

En el listado de repositorios disponibles abrir el contenido del repositorio agregado y seleccionar el proyecto deseado presionando click derecho en el proyecto y dando click a la opción checkout

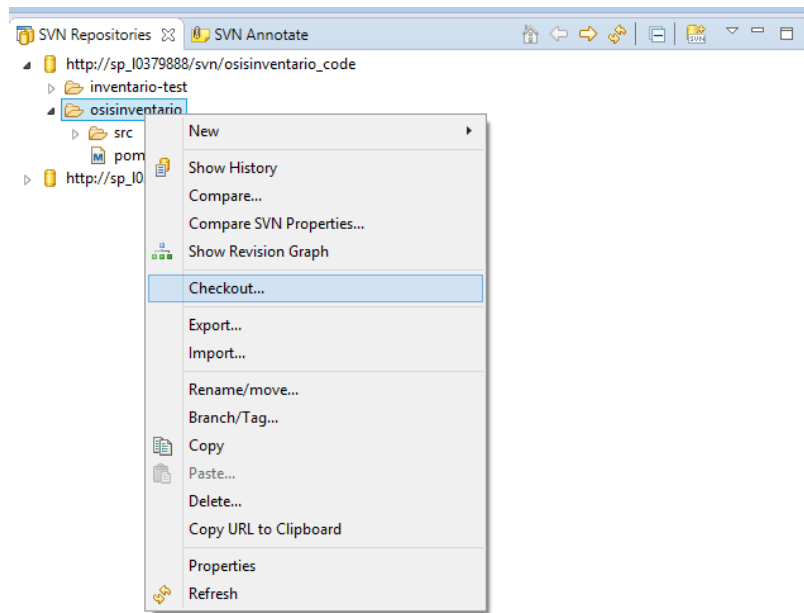


Ilustración 33 Checkout código

En la pantalla de configuración seleccionar la opción Check out as a Project in the workspace y en Finish.

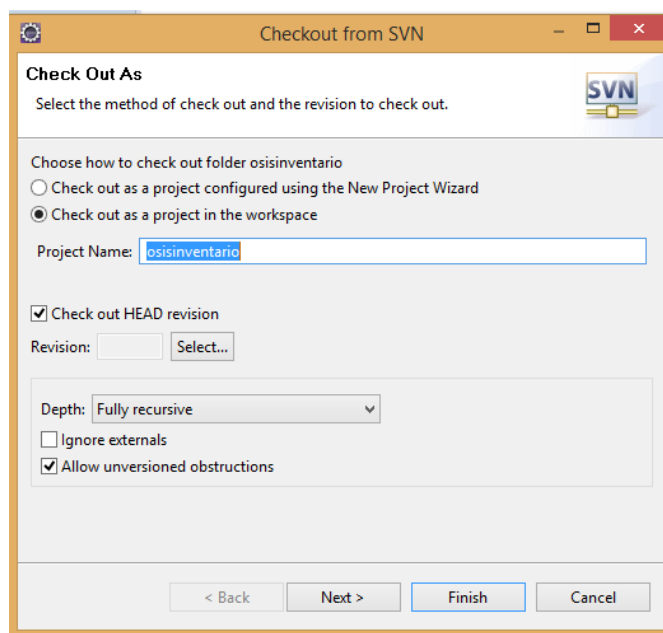


Ilustración 34 Check out in workspace

9.1.3 Convertir a Maven Project

Cuando descargamos un proyecto desde el SVN eclipse no lo reconoce automáticamente como un proyecto Maven para eso tenemos que dar click derecho al proyecto ya descargado en nuestro workspace y seleccionar Configure -> Convert to Maven Project

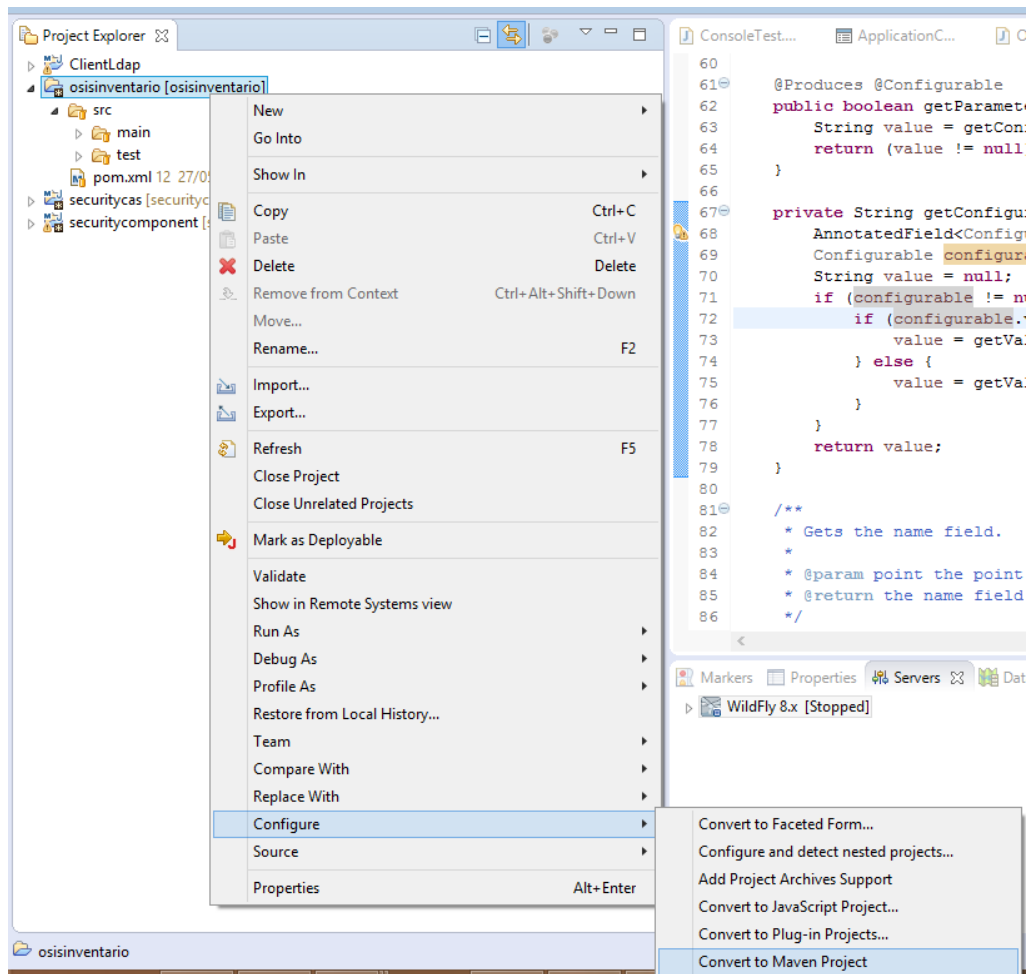


Ilustración 35 Convertir a Maven

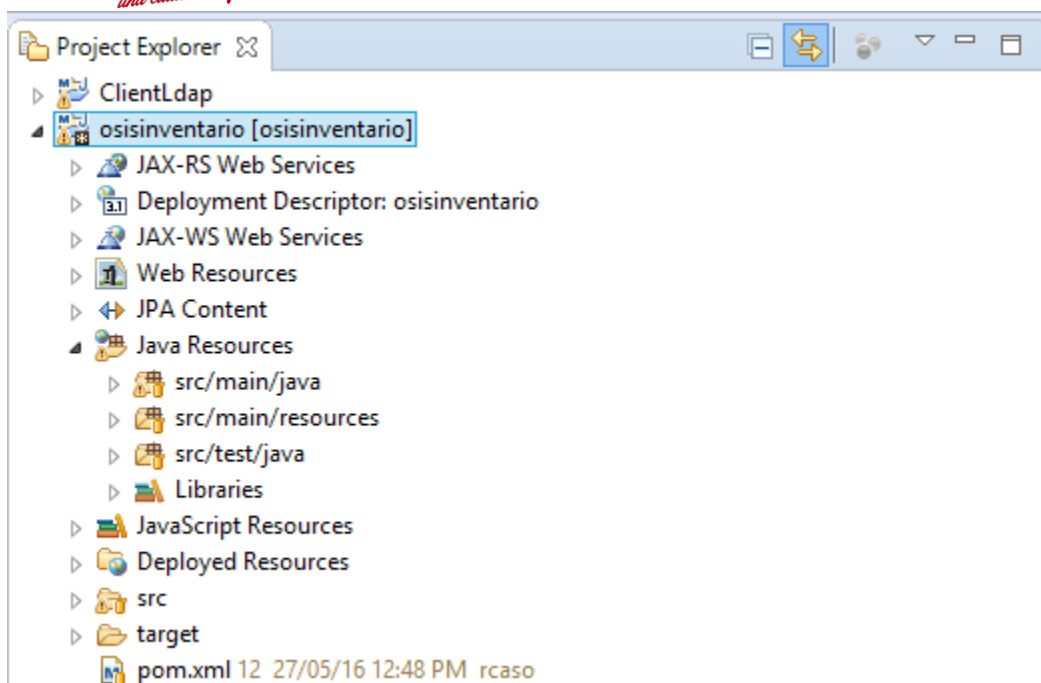


Ilustración 36 Proyecto reconocido como Maven

9.2 Crear nuevos Proyectos

Para crear nuevo proyectos se tomaran como base los archetypes creados para estandarizar la estructura de los proyectos.

9.2.1 Crea Nuevo Proyecto Back End

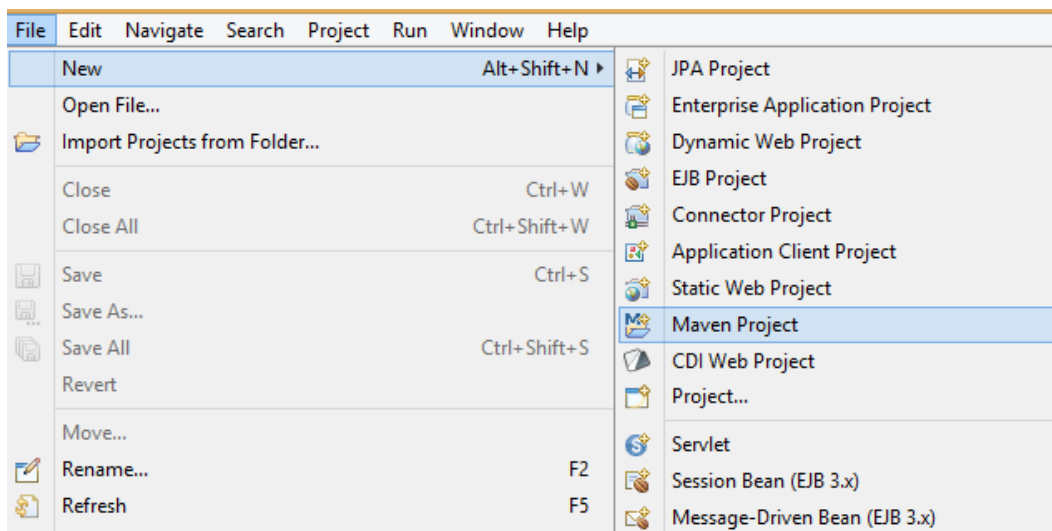


Ilustración 37 Click nuevo proyecto

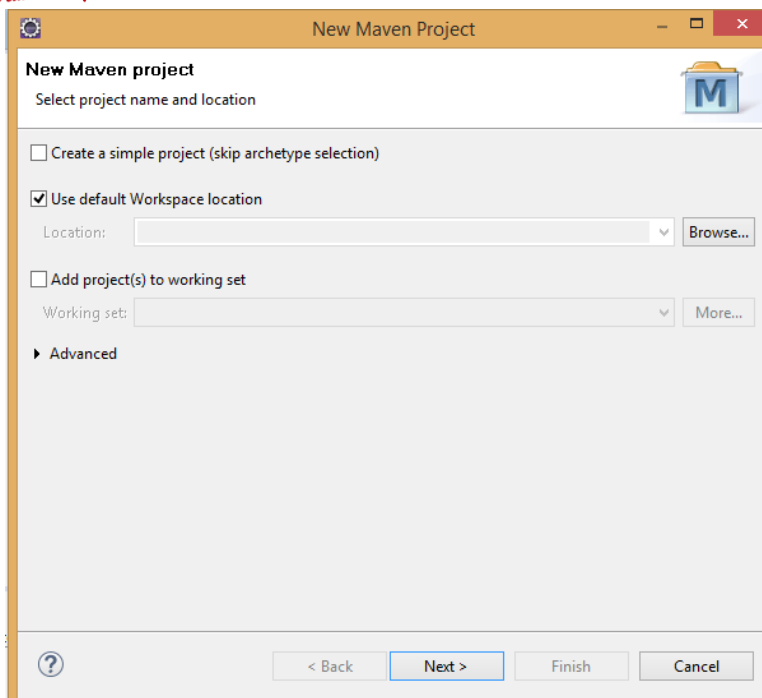


Ilustración 38 Utilizar archetypes

Escoger el catalogo remoto creado previamente en el combo y seleccionar el artifact mpfn-backendapp-archetype

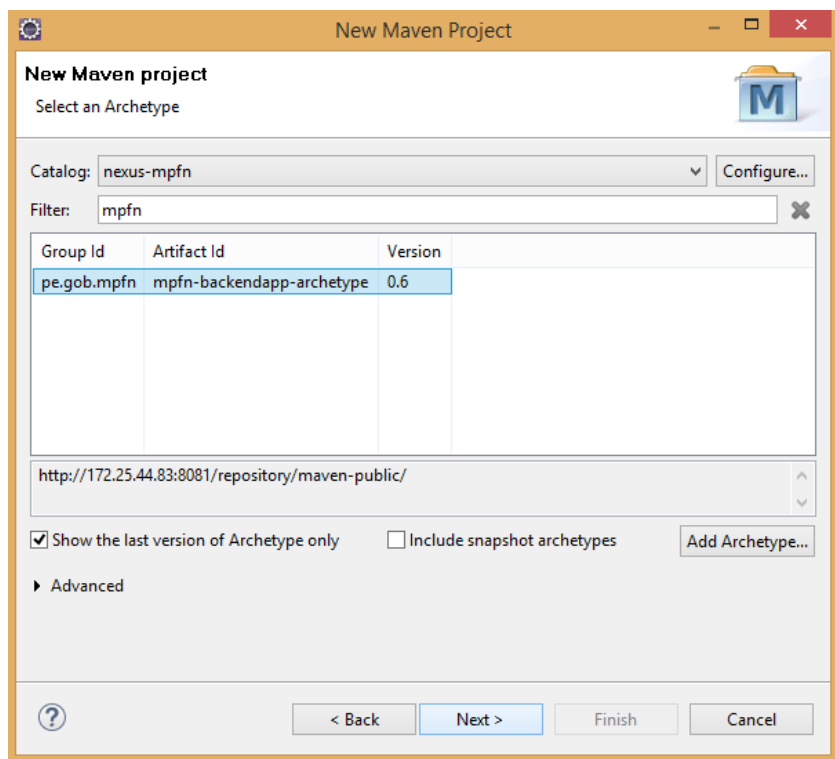


Ilustración 39 Seleccionar el archetype

Ingresa en Artifact Id: el nombre del proyecto web (de preferencia en minúsculas) y se autocompletará el nombre de paquetes base para el proyecto

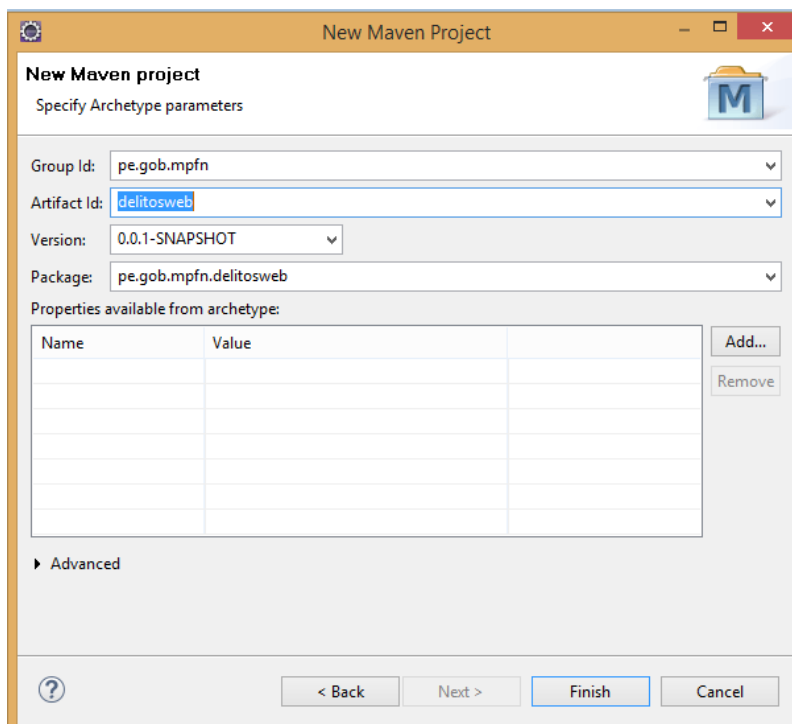


Ilustración 40 Configuración nombre de proyectos

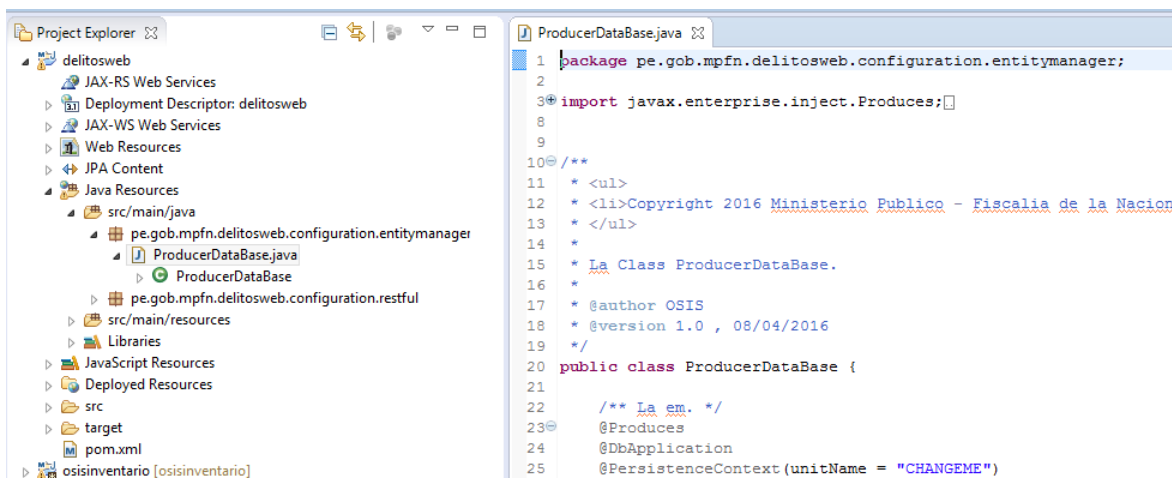


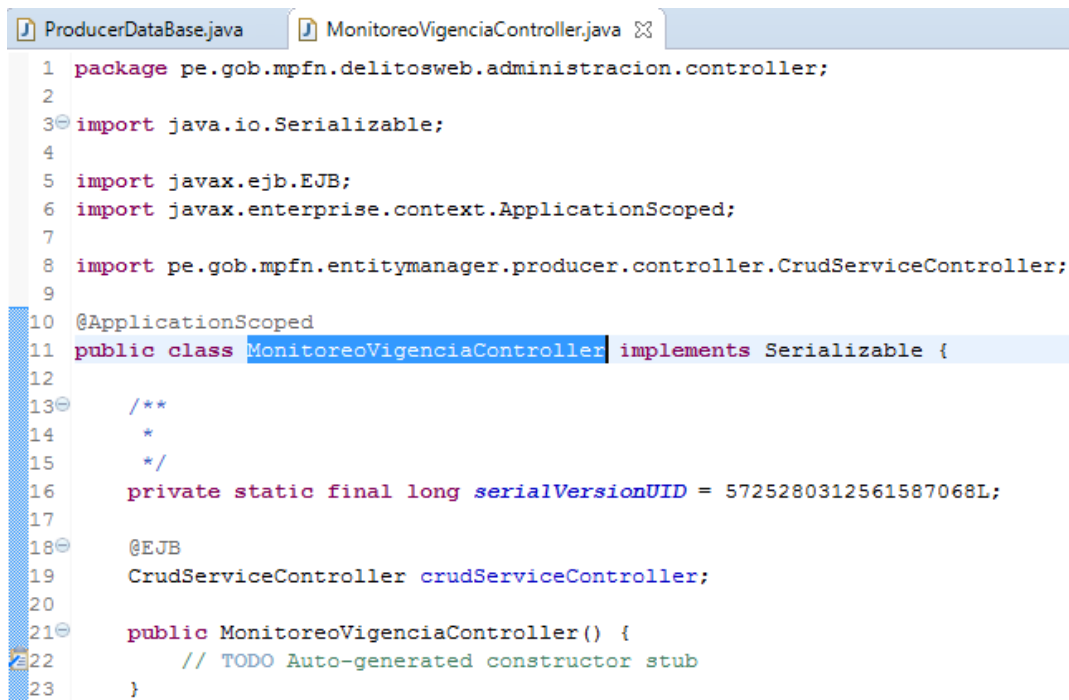
Ilustración 41 Proyecto Creado

9.3 Documentación de Clases

Para la documentación base de las clases se utilizara el plugin Jautodoc instalado previamente y configurado con la plantilla propuesta.

9.3.1 JAutodoc

Al crear una clase llamada por ejemplo MonitoreoVigenciaController se creara sin ninguna documentación estándar.



```
1 package pe.gob.mpfm.delitosweb.administracion.controller;
2
3 import java.io.Serializable;
4
5 import javax.ejb.EJB;
6 import javax.enterprise.context.ApplicationScoped;
7
8 import pe.gob.mpfm.entitymanager.producer.controller.CrudServiceController;
9
10 @ApplicationScoped
11 public class MonitoreoVigenciaController implements Serializable {
12
13     /**
14      *
15      */
16     private static final long serialVersionUID = 5725280312561587068L;
17
18     @EJB
19     CrudServiceController crudServiceController;
20
21     public MonitoreoVigenciaController() {
22         // TODO Auto-generated constructor stub
23     }
```

Ilustración 42 Crear nueva clase

Ubicar el cursor en la cabecera de la clase luego hacer un click con el botón derecho y seleccionamos la opción JAutodoc -> Add Javadoc

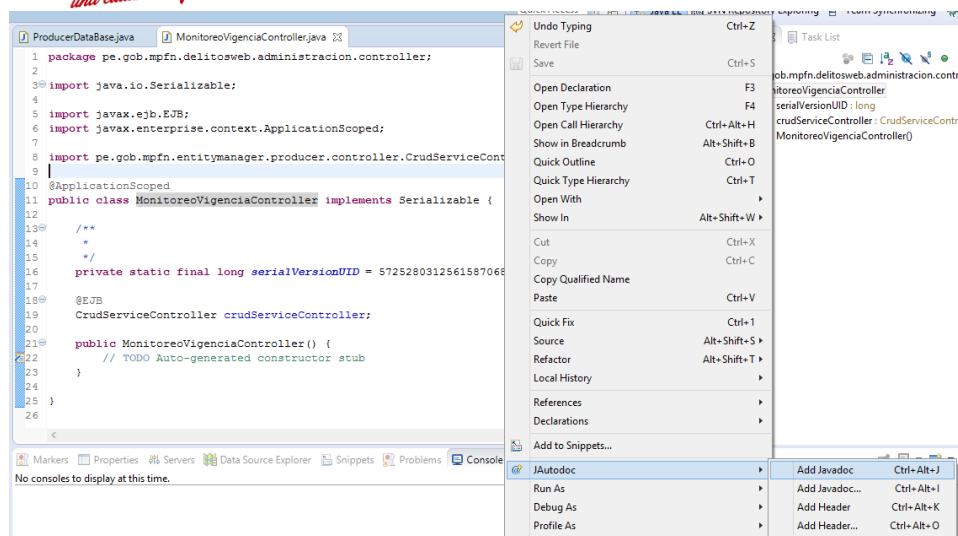


Ilustración 43 Agregar Comentarios

Se agregará la cabecera de comentario estándar de cada clase.



Ilustración 44 Cabecera estándar de comentario

9.4 Trabajar con SVN en los Proyectos

La herramienta para controlar las versiones del código fuentes de los proyectos será SVN para cada proyecto con sus componentes relacionados (más de un proyecto por aplicativo) deberán estar creados y sincronizados en su respectivo repositorio.

9.4.1 Subir un proyecto nuevo al SVN

Seleccionar el proyecto nuevo hacer un click derecho sobre él y seleccionar las opciones Team -> Share Project

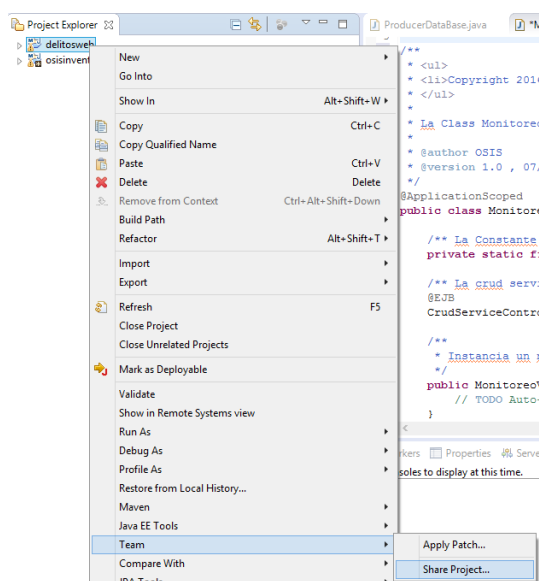


Ilustración 45 Compartir proyecto

Seleccionar SVN y Siguiendo

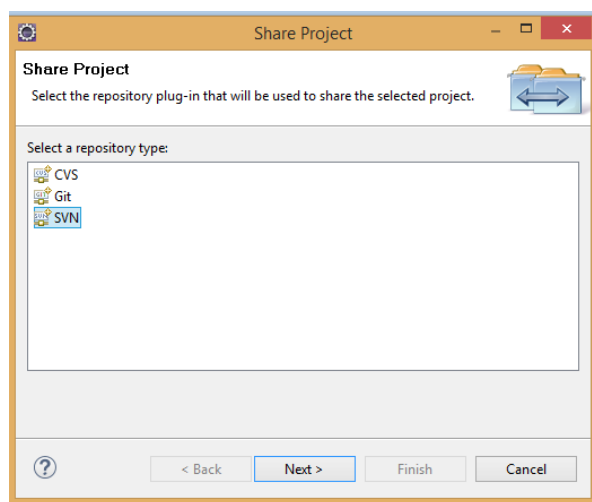


Ilustración 46 Seleccionar SVN

Seleccionar Create new repository location y Siguiente

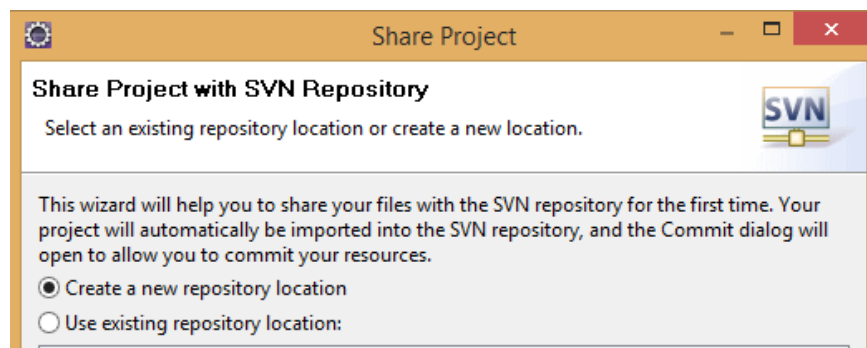


Ilustración 47 Configurar nuevo repositorio

Escribir la ruta url del repositorio SVN donde se subirá el código y Siguiente

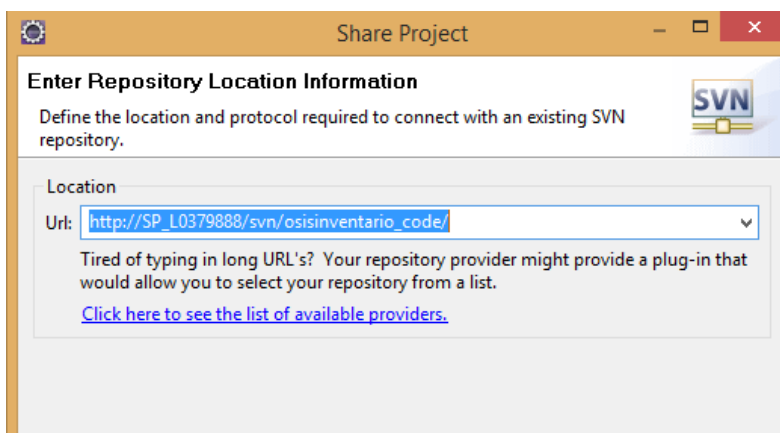


Ilustración 48 Repositorio donde se subirá el código

Seleccionar Use Project name as folder name y Siguiente

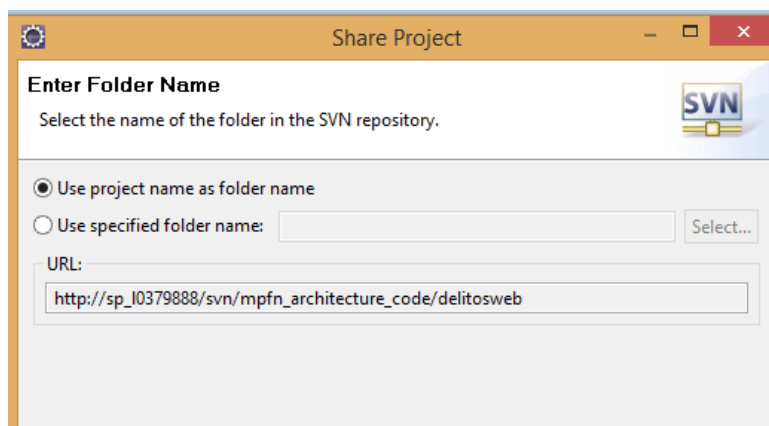


Ilustración 49 Nombre de proyecto

Colocamos un comentario del motive de la subida de código inicial al repositorio local

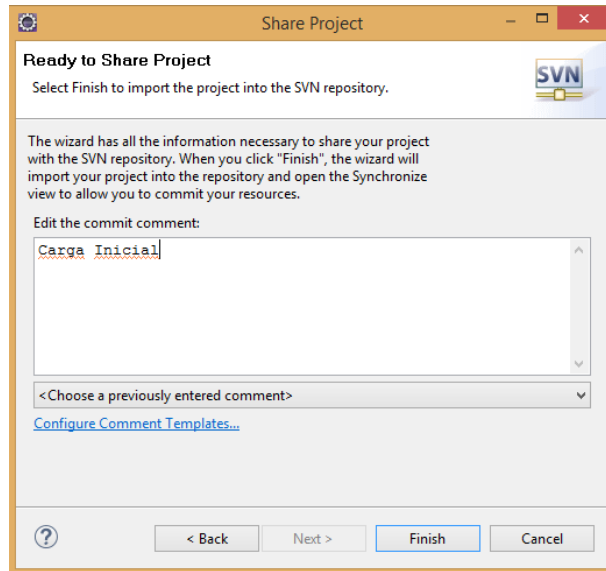


Ilustración 50 Añadir al repositorio local

Se nos abrirá la perspectiva de Team Synchronizing y seleccionamos solo el código fuente del proyecto es decir no debemos seleccionar lo siguiente .settings, .classpath, .project ni archivos dentro de la carpeta target

Damos click derecho sobre la selección y seleccionamos commit

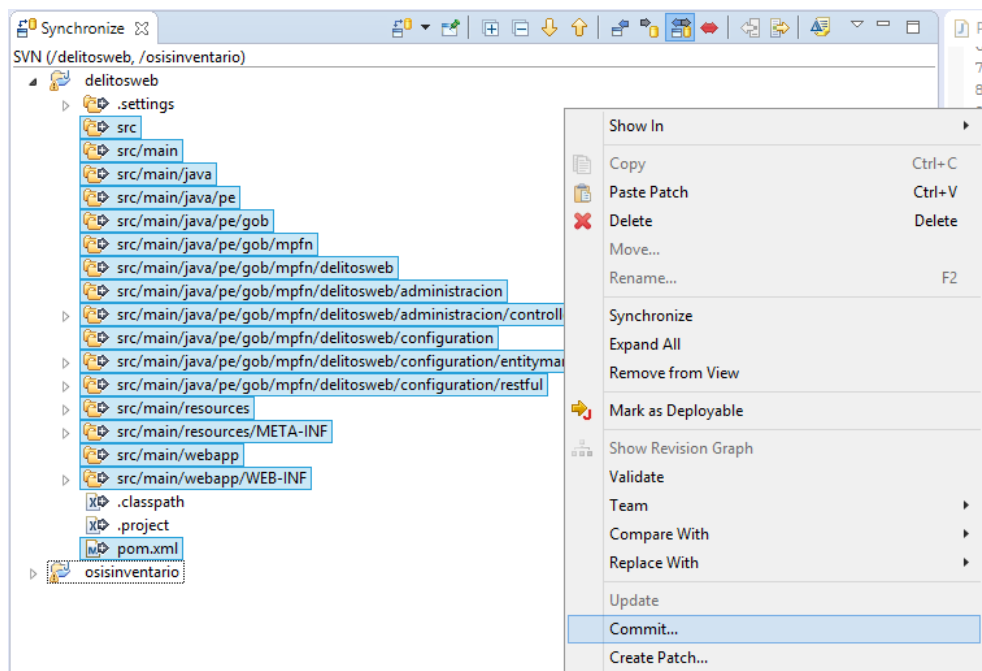


Ilustración 51 Seleccionar archivos para commit

Colocamos el comentario obligatorio del motivo de la subida de archivos al repositorio SVN.

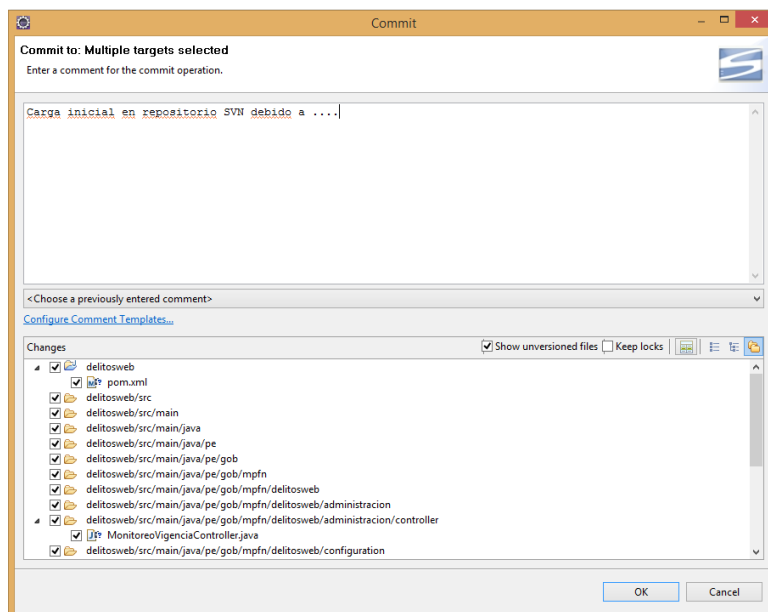


Ilustración 52 Confirmación y comentario del commit

En la perspectiva Java EE se ve el número de versión de los archivos

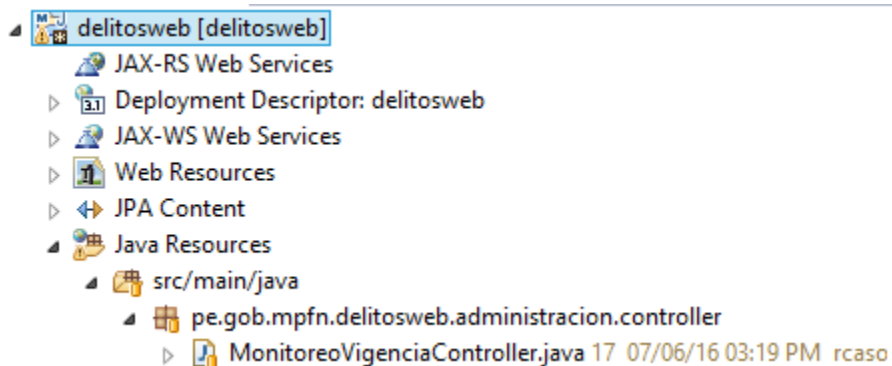


Ilustración 53 Número de versión de archivos

9.4.2 Subir Código al SVN

Para subir código solo se deben seleccionar las clases, archivos o paquetes modificados y dar un click derecho y seleccionar Team -> Commit

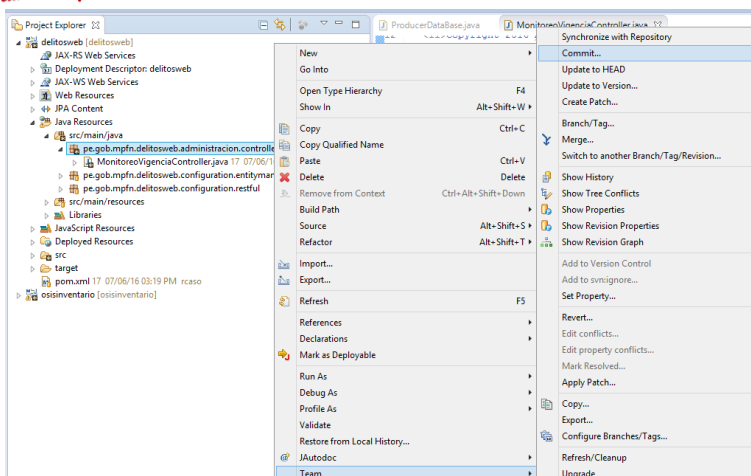


Ilustración 54 Commit modificaciones

Agregar el comentario obligatorio de la modificación de las fuentes

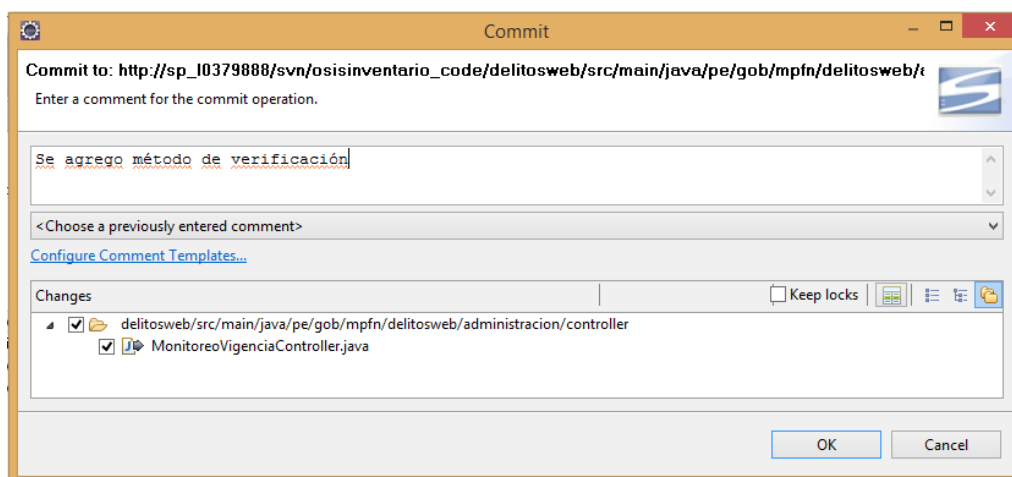


Ilustración 55 Especificar el motivo de la modificación y confirmar

Nueva y versión y ya no se muestra indicador de modificación en el archivo

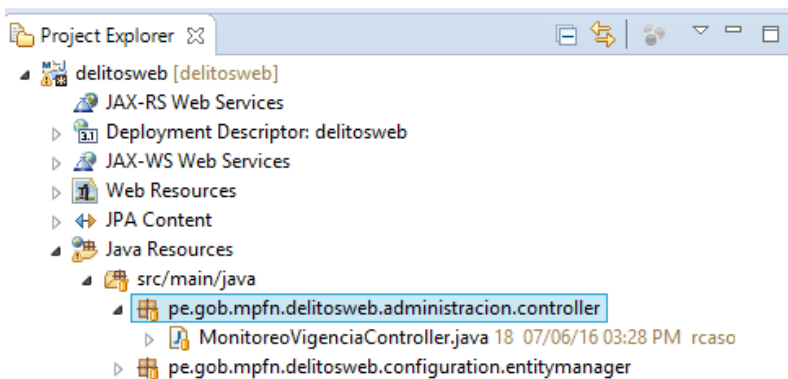


Ilustración 56 Nueva versión después del commit

9.4.3 Actualizar Código del SVN

Verificar que en la opción del eclipse Windows -> Preferences -> Team -> SVN -> Update to HEAD todas las opciones estén en **Prompt me for each conflict and let me decide**

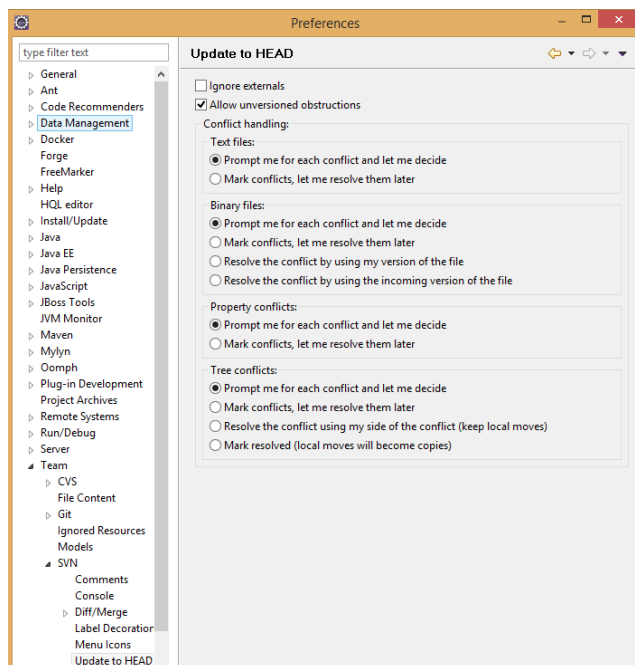


Ilustración 57 Configuración de conflictos

Seleccionar una porción del código o todo el proyecto que se desea actualizar y dar click derecho y seleccionar Team -> Update to HEAD

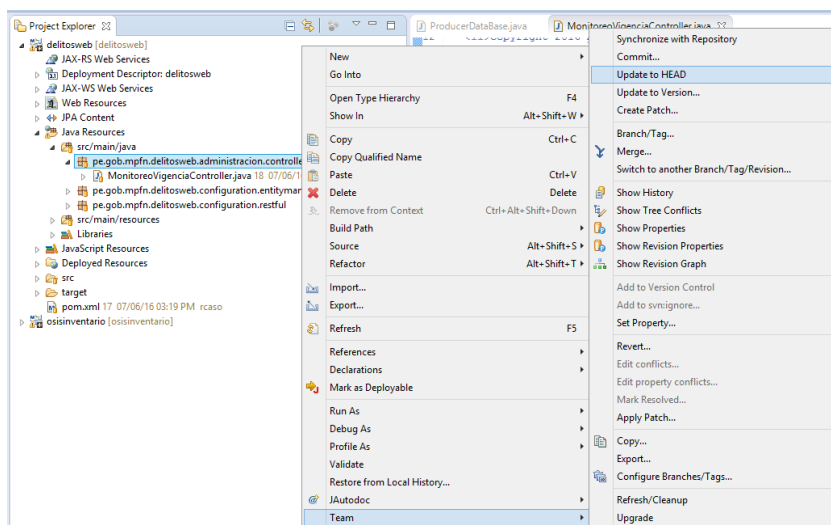


Ilustración 58 Actualizar porción del código