Capítulo 4:

DESARROLLO.

4.1 Implementación segura de objetos de Base de datos.

Diseño Seguro de Bases de Datos en PostgreSQL.

Todo DBMS tiene la obligación de facilitar mecanismos para prevenir los fallos (subsistema de control), detectarlos una vez que se han producido (subsistema de detección) y corregirlos después de haber sido detectados (subsistema de recuperación). Con esto el DBMS debe garantizar que se cumplan los tres principios de seguridad, los cuales son integridad, disponibilidad y confidencialidad, para lo que PostgreSQL cumple con estos principios.

La primera de dichas características es la integridad, la cual se garantiza no sólo al momento de crear la base de datos por medio del SQL en PostgreSQL, si no que iendo un poco más atrás, al análisis y diseño de la base de datos. Un buen diseño de la base de datos ayudará a conservar la integridad de la información dentro del DBMS.

Desde el diseño debemos definir el cumplimiento de los requerimientos que haga la organización, la integridad radica en asegurar que los datos almacenados sean correctos, es decir verídicos, además de asegurar que lo que se trata de hacer es correcto. Por ejemplo, comúnmente tenemos el diseño de una entidad de la siguiente forma:

Figura 4.1.1 Estructura de una tabla.

Por lo general no hay preocupación porque la aplicación que manipula los datos y los objetos de la base, lo hace de forma adecuada.

Los datos deben estar protegidos contra modificaciones accidentales o maliciosas, considerando incluso la inserción de datos falsos o la destrucción de los mismos. Dichas verificaciones deben contemplar desde los inicios del ciclo de vida de las bases de datos. Si en un principio se elabora un diseño a la ligera, la seguridad de la base de datos se verá comprometida.

Tipos de Integridad protegidos mediante los siguientes tipos de restricción:

Integridad de la tabla.
· Llave Primaria (Primary Key).
· Valor único (UNIQUE).
Integridad de dominio.
· Verificar valor NULL.
· Valor por default.
· Verificación de valor (Constraint de tipo Check).
Integridad referencial.
· Llave Foránea (Foreign Key).

Integridad de Tabla. Cada una de las tablas que tenemos dentro del DBMS debe de tener asignada una llave primaria, es decir un campo que garantice el registro único, con ello evitamos la redundancia.

Integridad de Dominio: Es muy importante que en la base de datos almacenemos exactamente lo deseado. Por medio de cláusulas SQL como Constraints de tipo Check, valores nulos y valores por default.

Integridad referencial: Este tipo de integridad garantiza que los registros se integren de forma adecuada en las tablas, sin violar el principio básico de las bases de datos relaciónales.

Un ejemplo creado con SQL para mostrar los principios básicos de la creación de una tabla es el siguiente:

```
CREATE TABLE becario(
rfc VARCHAR(10) NOT NULL PRIMARY KEY ,
nombre VARCHAR(45) NOT NULL check(trim(nombre) <> ''),
apellido VARCHAR(45) NOT NULL check(trim(apellido) <> ''),
direccion VARCHAR(50) NULL check(trim(direccion) <> ''),
fechadeingreso timestamp not null default now() check (fechaingreso > '1994-01-01'),
emailunidad VARCHAR(50) NOT NULL check( correo ~ '[a-z0-9_]+@[a-z0-9_](.?[a-z0-9_]+)*') unique,
promedio float not null default 0 check(promedio >= 0 and promedio =< 10)
);
```

En la creación anterior de la tabla se contemplan las siguientes cláusulas:
PRIMARY KEY: Identificador único del registro.
NOT NULL: Nos obliga a ingresar algo en el campo.
NULL: el campo puede ser nulo.
DEFAULT: Es un valor por default en caso de no poner uno.
CHECK: realiza validaciones de dominio, por ejemplo una expresión regular para el correo electrónico verifica que no se ingresen espacios en blanco.
A veces esta parte es muy descuidada por los analistas y/o diseñadores de la base de datos, dejando la tarea de validar este tipo de cosas al desarrollador de alguna aplicación que explota los datos, por lo que no es recomendable hacerlo.

Al hacer un buen diseño de la base de datos y especificar los tipos de datos y sus dominios, PostgreSQL es capaz de garantizar que en la Base de Datos pueda controlarse la redundancia a su mínima expresión, manteniendo consistente la base de datos y así conseguir la integridad de los datos que se persigue en un inicio. Parte importante de la Seguridad en Base de datos es hacer estas validaciones desde el RDBMS.

PostgreSQL puede echar mano además de las cláusulas básicas de creación de las bases de datos, de otra característica que tiene, los Trigger los cuales son programas que se disparan en cuanto se realiza una cierta acción, son de gran utilidad cuando queremos evitar que la acción realizada ponga en riesgo la integridad de los datos o para identificar posibles problemas. Por ejemplo, la modificación inapropiada de datos por parte de algún usuario.

Una vez garantizado o al menos disminuido el riesgo de que la base de datos pierda su integridad, se debe evaluar el tipo de datos almacenados y quienes van a acceder a determinada información producida para que la base de datos no pierda su disponibilidad.

Para mantener la disponibilidad de un servidor de base de datos se tiene un principio básico: la redundancia física. En él se apoya la recuperación de los datos ante cualquier fallo. Lo importante ante cualquier tipo de fallo es asegurar que, después de una actualización, la BD se quede en un estado consistente. Por definición, la Base de Datos se encuentra en este estado antes de que se ejecute una transacción y deberá recobrarlo al concluir ésta.

Las propiedades principales que debe poseer una transacción son:

- Atomicidad.
- Preservación de la consistencia.
- Aislamiento. Una transacción no muestra los cambios que produce hasta que finaliza.
- Persistencia. Una vez que la transacción finaliza con éxito, sus efectos perduran en la BD.

Las sentencias de que consta la transacción deberán, por tanto, deshacerse (rollback) en caso de no ser concluidas satisfactoriamente. De darse este caso, las actualizaciones de que consta la transacción se graban (commit) si terminó con éxito. Esto se puede implementar como medida de seguridad para la consistencia de la base de datos. PostgreSQL soporta transacciones clásicas mediante las sentencias COMMIT y ROLLBACK.

Sin duda, una parte fundamental para mantener disponible la información es respaldar los datos que se encuentran en la base de datos. Para ello PostgreSQL permite realizar dos tipos de respaldos, los cuales se deben adaptar de acuerdo a las políticas de respaldo implementadas por la organización. Estos tipos de respaldos son llamados comúnmente respaldos en frío y caliente. El primero hace una copia de las estructuras físicas de las bases de datos cuando éstas no estén disponibles a los usuarios. La copia de los datos se hace a través de las utilerías del DBMS o comandos del Sistema Operativo tales como tar, cp, cpio,etc. Mientras, el otro se realiza cuando la BD está abierta y funciona el mecanismo de LOG. Consiste en copiar todos los archivos de LOG correspondientes a un directorio determinado.

De acuerdo a las necesidades de cada una de las organizaciones se debe elegir un tipo de respaldo que desea realizarse. Para poder escoger el que más adecuado a las necesidades de la organización, y elegirlo correctamente, debemos primero conocer cuáles son los tipos de respaldos que ofrece PostgreSQL.

Persistencia de objetos utilizando java.

INTRODUCCIÓN.

Para la mayoría de las aplicaciones, almacenar y recuperar información implica alguna forma de interacción con una base de datos relacional. Esto ha representado un problema fundamental para los desarrolladores porque algunas veces el diseño de datos relacionales y los ejemplares orientados a objetos comparten estructuras de relaciones muy diferentes dentro de sus respectivos entornos. Las bases de datos relacionales están estructuradas en una configuración tabular y los ejemplares orientados a objetos normalmente están relacionados en forma de árbol. Esta 'diferencia de impedancia' ha llevado a los desarrolladores de varias tecnologías de persistencia de objetos a intentar construir un puente entre el mundo relacional y el mundo orientado a objetos.

Persistir objetos Java en una base de datos relacional es un reto único que implica serializar un árbol de objetos Java en una base de datos de estructura tabular y viceversa. Este reto actualmente está siendo corregido por varias herramientas diferentes. La mayoría de estas herramientas permite a los desarrolladores instruir a los motores de persistencia de cómo convertir objetos Java a columnas/registros de la base de datos y al revés. Esencial para este esfuerzo es la necesidad de mapear objetos Java a columnas y registros de la base de datos de una forma optimizada en velocidad y eficiencia.

PERSISTENCIA MEDIANTE SERIALIZACIÓN.

La serialización de un objeto consiste en obtener una secuencia de bytes que represente el estado de dicho objeto. Esta secuencia puede utilizarse de varias maneras (puede enviarse a través de la red, guardarse en un fichero para su uso posterior, utilizarse para recomponer el objeto original, etc.).

ESTADO DE UN OBJETO.

El estado de un objeto viene dado, básicamente, por el estado de sus campos. Así, serializar un objeto consiste, básicamente, en guardar el estado de sus campos. Si el objeto a serializar tiene campos que a su vez son objetos, habrá que serializarlos primero. Éste es un proceso recursivo que implica la serialización de todo un grafo (en realidad, un árbol) de objetos. Además, también se almacena información relativa a dicho árbol, para poder llevar a cabo la reconstrucción del objeto serializado.

INTERFAZ SERIALIZABLE.

Para que un objeto sea serializable, debe implementar la interfaz java.io.Serializable. Esta interfaz no define ningún método. Simplemente se usa para 'marcar' aquellas clases cuyas instancias pueden ser convertidas a secuencias de bytes (y posteriormente reconstruidas). Objetos tan comunes como String, Vector o ArrayList implementan Serializable, de modo que pueden ser serializados y reconstruidos más tarde. Para serializar un objeto no hay más que declarar el objeto como serializable: public class MiClase implements java.io.Serializable El sistema de ejecución de Java se encarga de hacer la serialización de forma automática.

Ejemplo.
Almacenamiento de objetos en variables de sesión.
Es posible utilizar los mecanismos de serialización disponibles para serializar un objeto guardándolo en una sesión y para realizar el proceso inverso, recuperándolo desde la sesión.
Primer paso: El objeto deberá implementar la interfaz java.io.Serializable.

```
package classes;
import java.io.Serializable;
public class Integrante implements Serializable{
```

```
	/**
	 *
	 */
	private static final long serialVersionUID = 1L;
	/**
	 * @uml.property  name="nombre"
	 */
	private String nombre;
	/**
	 * @uml.property  name="rol"
	 */
	private String rol;
	/**
	 * @param  nombre
	 * @uml.property  name="nombre"
	 */
	public void setNombre(String nombre){
		this.nombre=nombre;
	}
	/**
	 * @param  rol
	 * @uml.property  name="rol"
	 */
	public void setRol(String rol){
		this.rol=rol;
	}
	/**
	 * @return
	 * @uml.property  name="nombre"
	 */
	public String getNombre(){
		return this.nombre;
	}
	/**
	 * @return
	 * @uml.property  name="rol"
	 */
	public String getRol(){
		return this.rol;
	}
}
```

Segundo paso: El objeto se guardará en una variable de sesión para conservar su estado mientras el usuario navegue en el portal.

```
//Recuperando los datos que deberá persistir el objeto.
String nombre=request.getParameter("nombre");
String rol=request.getParameter("rol");

//Insertando los datos en el arreglo de objetos.
HashMap<String, Integrante> nombres= new HashMap<String, Integrante>();
StringTokenizer st = new StringTokenizer(nombre, ",");
String rfc=st.nextToken();
String nom=st.nextToken();
Integrante in=new Integrante();
in.setNombre(nom);
```

```
in.setRol(rol);
nombres.put(rfc, in);

//Guardando los datos en la variable de sesión que permitirá la persistencia.
HttpSession siceb = request.getSession();
siceb.setAttribute("integrantes", nombres);
```

Tercer paso: Cuando el usuario haya hecho todos sus cambios, el arreglo de objetos se recuperará de la variable de sesión para guardarse en la base de datos.

```
//Recuperando el arreglo de objetos.
HttpSession siceb = request.getSession();
HashMap<String, Integrante> nombres= HashMap<String, Integrante>)siceb.getAttribute("integrantes");

//Método que inserta el arreglo en la base de datos.
public static int inserta(HashMap<String,Integrante> nombres, int idproyecto){
        String query="Insert into becarioproyecto(rfc, idproyecto, tipoparticipante) VALUES(?,?,?);";
        Connection conn=null;
    PreparedStatement ps=null;
    int bandera=0;
        try{
            Set<Entry<String, Integrante>> set = nombres.entrySet();
          Iterator<Entry<String, Integrante>> i = set.iterator();
          conn=ConnectionFactory.getConnection();
              while( i.hasNext()) {
                  Map.Entry<String,Integrante> me = (Map.Entry<String,Integrante>)i.next();
                   Integrante prov = (Integrante)me.getValue();
                    ps =conn.prepareStatement(query);
                  ps.clearParameters();
                  ps.setString(1, (String)me.getKey());
                  ps.setInt(2, idproyecto);
                  if(prov.getRol().equals("lider")) ps.setString(3, "l");
                  if(prov.getRol().equals("participante")) ps.setString(3, "p");
                  if(prov.getRol().equals("ocasional")) ps.setString(3, "o");
            bandera+= ps.executeUpdate();
              }
        }
    catch (Exception e)
        {
          e.printStackTrace();
        }
        finally
        {
ConnectionFactory.close(ps);
ConnectionFactory.close(conn);
        }
        return bandera;
}

//Uso de dicho método para la inserción del arreglo en la base de datos.
Proyecto.inserta(nombres, idProyecto);
```

Ejemplo.
Obtención de objetos de la base de datos e insertarlo en variables de sesión.
Primer paso: Extracción de la información de la base de datos.

```
//Método que extrae el arreglo de objetos de la base de datos.
public static HashMap<String,Integrante> obtBecario(int idproyecto){
		HashMap<String, Integrante> datos= new HashMap<String, Integrante>();
		String rfc, nombre, rol, nombreC, ap;

		String query=null;
		query="Select becario.rfc, becarioproyecto.tipoparticipante, becario.nombre,
becario.apellido from becarioproyecto, becario where becarioproyecto.idproyecto="+idproyecto+"
AND becarioproyecto.rfc=becario.rfc;";
		Connection conn=null;
	ResultSet rs=null;
	PreparedStatement pstmt=null;
		try
			{
				conn=ConnectionFactory.getConnection();
			pstmt =conn.prepareStatement(query);
		rs = pstmt.executeQuery(); //se ejectuta y se guarda en rs
		while (rs.next())
		{
			rfc=rs.getString(1);
			rol=rs.getString(2);
			nombre=rs.getString(3);
			ap=rs.getString(4);
			nombreC= nombre + " " + ap;
			Integrante in=new Integrante();
			in.setNombre(nombreC);
			if(rol.equals("l"))in.setRol("lider");
			if(rol.equals("p"))in.setRol("participante");
			if(rol.equals("o"))in.setRol("ocasional");
			datos.put(rfc,in);

		}


			}
		catch (Exception e)
	{
	  e.printStackTrace();
	}
		finally
		{ ConnectionFactory.close(rs);
	ConnectionFactory.close(pstmt);
	ConnectionFactory.close(conn);
		}
		return datos;
}
```

Segundo paso: Persistencia del objeto en una variable de sesión.

```
//Guardado del arreglo de objetos en una variable de sesión.
HashMap<String, Integrante> hombres=Proyecto.obtBecario(idProyecto);
HttpSession siceb = request.getSession();
```

siceb.setAttribute("integrantes", hombres);

Tercer paso: Manipulación del arreglo de objetos para nuestra aplicación.

```
//Obteniendo el arreglo de la variable de sesión.
HttpSession siceb = request.getSession();
HashMap<String, Integrante> hombres= HashMap<String, Integrante>)siceb.getAttribute("integrantes");

//Iteración del arreglo para obtener el estado de los objetos.
Set set1 = hombres.entrySet();
Iterator in = set1.iterator();
int j=0;
while( in.hasNext()) {
        Map.Entry men = (Map.Entry)in.next();
        Integrante ingte = (Integrante)men.getValue();
        //Imprimiendo el rfc del becario.
        out.println(men.getKey());
        //Imprimiendo su nombre.
        out.println(ingte.getNombre());
        //Imprimiendo su rol en el proyecto.
        out.println(ingte.getRol());
}
```

4.2 Programación de los módulos.

Módulos funcionales de los cuales se compone el SIGEB.

Como se revisó en la sección de requerimientos el Sistema de Gestión de Becarios de la Unidad de Servicios de Cómputo Académico se compone de cuatro módulos funcionales que se describen a continuación:

- **Módulo 1: Evaluación de becarios.**
- **Módulo 2: Proyectos.**
- **Módulo 3: Ex becarios.**
- **Módulo 4: Reporte semestral de actividades.**

De los cuales se explicará su funcionamiento y la cobertura de los requerimientos mediante la arborescencia del proyecto Web.

HERRAMIENTA DE DESARROLLO UTILIZADA.

Para el desarrollo del SIGEB se utilizó la herramienta de desarrollo de aplicaciones Java Eclipse, porque es una herramienta muy completa en cuanto a la programación Web, además de ser portable, no consume muchos recursos de la computadora, tiene Licencia Pública General de GNU (GNU GPL) y cuenta con extensiones (plugins) para diferentes tipos de necesidades de programación aunque no todas son de libre distribución.

Primero se tuvo que descargar la versión para desarrolladores de Java Enterprise Edition de Eclipse de la siguiente página: http://www.eclipse.org/downloads/. Esta versión es necesaria porque dicha versión contiene los editores adecuados para la programación Java-WEB.

Figura 4.2.1 Versión de Eclipse a descargar tal y como se muestra en la página de descargas.

Después de la descarga no es necesario hacer ninguna instalación pero como prerrequisito indispensable para el correcto funcionamiento de Eclipse es necesario tener el JDK de Java instalado en la máquina, y para fines del sistema también es necesario tener instalado el motor para servlets Tomcat, para su instalación véase el apartado 3.5 de esta tesis.

Al ejecutar el archivo de eclipse de la carpeta descomprimida lo primero que solicitará eclipse es una dirección para guardar los proyectos denominada workspace.

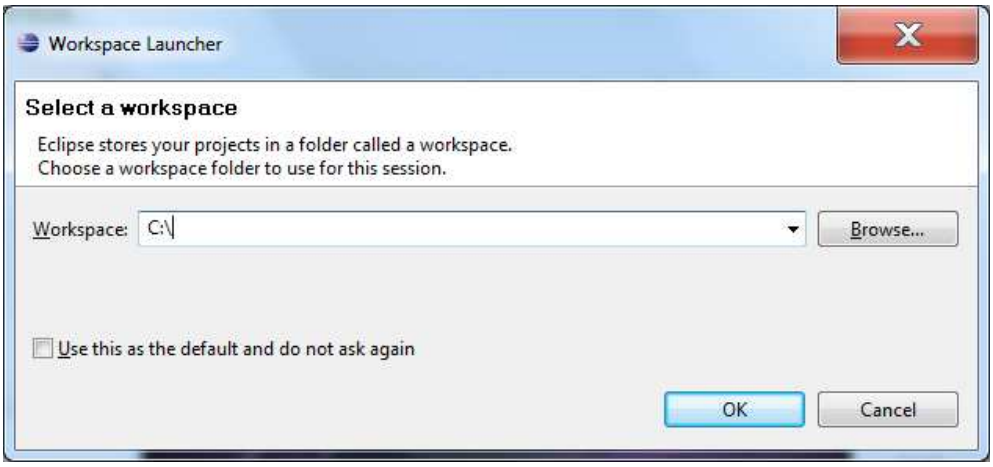


Figura 4.2.2 Asistente para la selección del directorio de trabajo.

Después de haber especificado la ruta para el worspace, eclipse habrá arrancado con el siguiente entorno de trabajo.

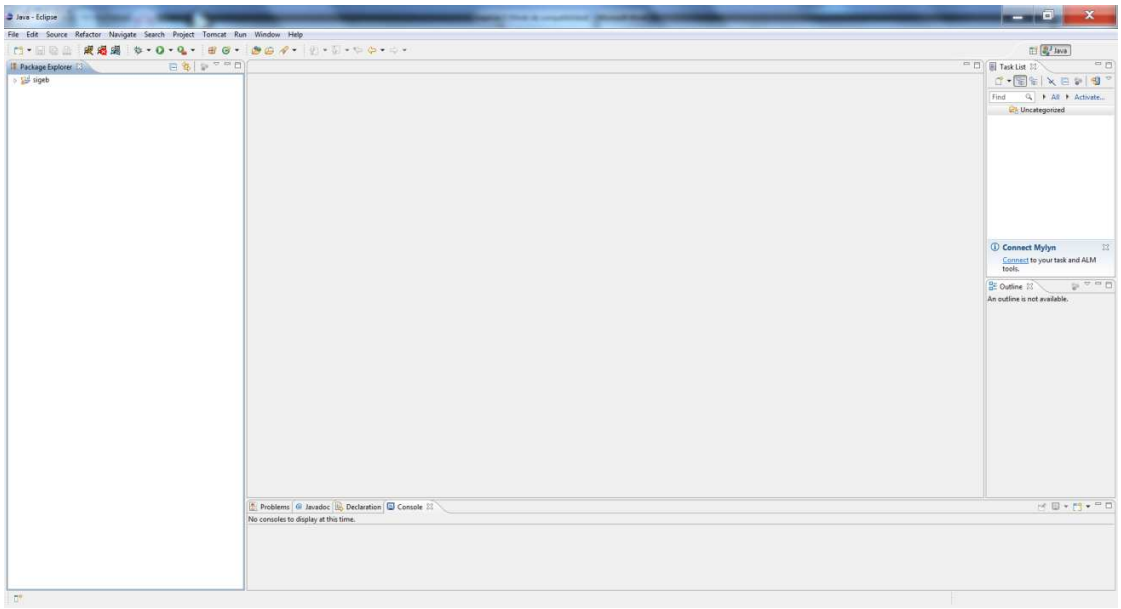


Figura 4.2.3 Vista general del entorno de trabajo en Eclipse.

Como mínimo para empezar a trabajar en el SIGEB se tienen que conocer las siguientes secciones del entorno de trabajo.

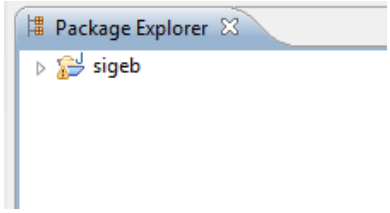


Figura 4.2.4 Sección que abarca la estructura de archivos de un proyecto de Eclipse.

Ésta sección se refiere a la estructura de archivos en el proyecto con el que se esté trabajando en Eclipse.

Figura 4.2.5 Sección que abarca los editores para archivos en Eclipse.

Aquí es donde se mostrarán los archivos que estemos editando, siempre y cuando sean soportados por Eclipse, en nuestro caso soportará todas las extensiones .java, .jsp, .xml, .html, .txt necesarias para el SIGEB.

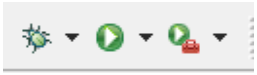


Figura 4.2.6 Sección que abarca los botones de ejecución de aplicaciones en Eclipse.

Esta sección será necesaria para ejecutar arbitrariamente las aplicaciones Java, siempre y cuando tengan un método main.

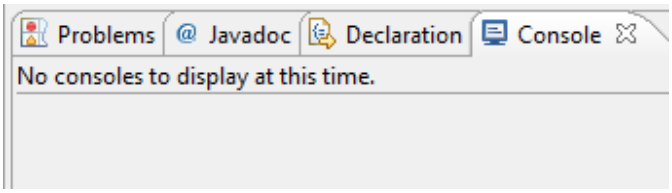


Figura 4.2.7 Sección que abarca la salida estándar de consola Java en Eclipse.

Esta sección corresponde a las diferentes salidas que nos puede dar nuestra aplicación, por ejemplo la salida en consola de la Máquina Virtual de Java o los diferentes errores o advertencias en tiempo de ejecución que puedan ocurrir.

Ahora lo que sigue es la configuración de Tomcat en Eclipse y la creación del proyecto web.

Primero se tiene que descargar el plugin de Tomcat para Eclipse de la siguiente página: http://www.eclipse-plugins.info/eclipse/index.jsp después descomprimir el archivo descargado en la carpeta de dropins de la raíz de Eclipse que se descargó anteriormente.

Después de haber reiniciado Eclipse se configurará en la pestaña de Window -> Preferences para que pueda reconocer los archivos de Tomcat.

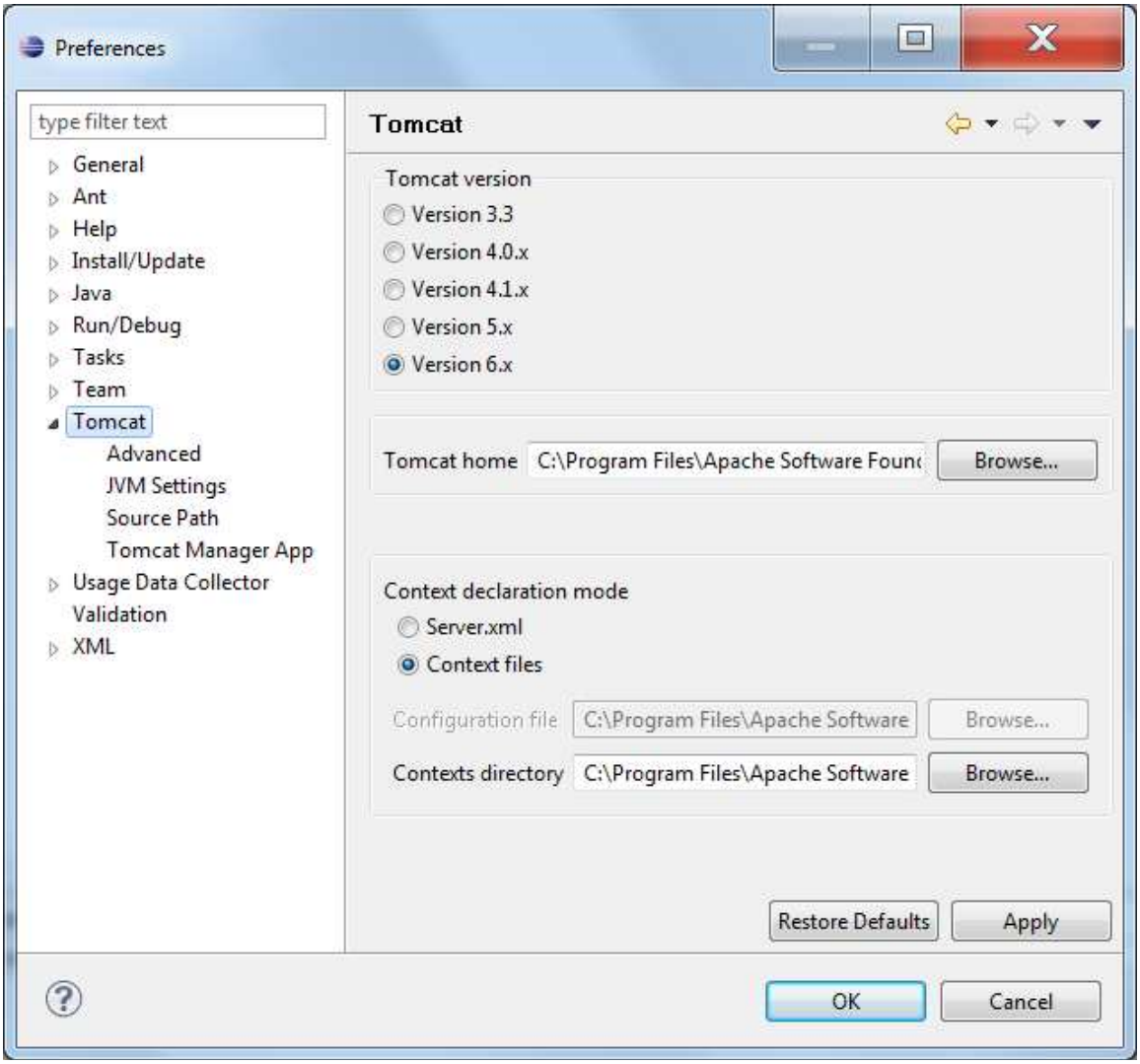


Figura 4.2.8 Asistente de configuración de Tomcat en Eclipse.

Habiendo aparecido la siguiente ventana se procedderá a seleccionar la liga a Tomcat de la parte derecha de la ventana, una vez hecho esto en la sección de “Tomcat vesion” se seleccionará la versión de Tomcat que se tenga instalada en la máquina de desarrollo.

Después de ello habrá que seleccionar la ubicación de la raíz de Tomcat, es decir en donde se ubican los archivos que se instalaron en la computadora cuando se instaló Tomcat.

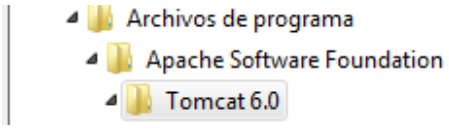


Figura 4.2.9 Ruta más común para la instalación de Tomcat en sistemas Windows.

Finalizada esta parte ya no será necesario hacer más configuraciones, ahora los siguientes botones habrán sido habilitados para la manipulación del servidor local desde Eclipse.

Figura 4.2.10 Botones de manipulación del servidor Tomcat en Eclipse.

La funcionalidad de los botones de izquierda a derecha es la siguiente: Arrancar Tomcat, detener Tomcat y reiniciar Tomcat.

Para comprobar la correcta configuración de Eclipse basta con arrancar el servidor Tomcat con el botón antes mencionado para que se obtenga una salida en consola similar a la siguiente.

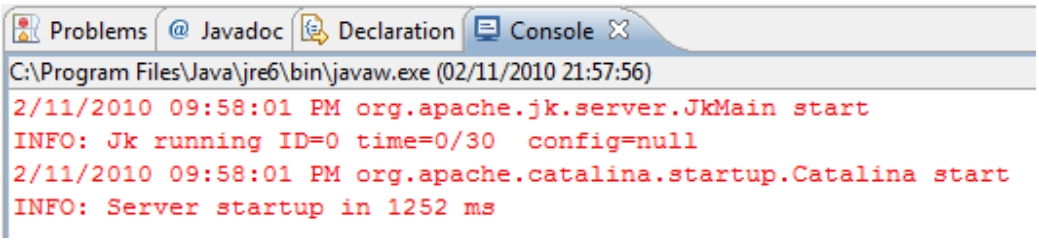


Figura 4.2.11 Salida esperada cuando se arranca el servidor Tomcat desde Eclipse.

Lo siguiente en este momento será crear un proyecto de Tomcat que permitirá trabajar y desarrollar bajo la plataforma de Java.-Web.

Primero hay que ir a la pestaña de File->New->Project.
Después desplegará todos los proyectos que soporta Eclipse pero para fines del sistema será un proyecto Tomcat como se muestra la imagen.

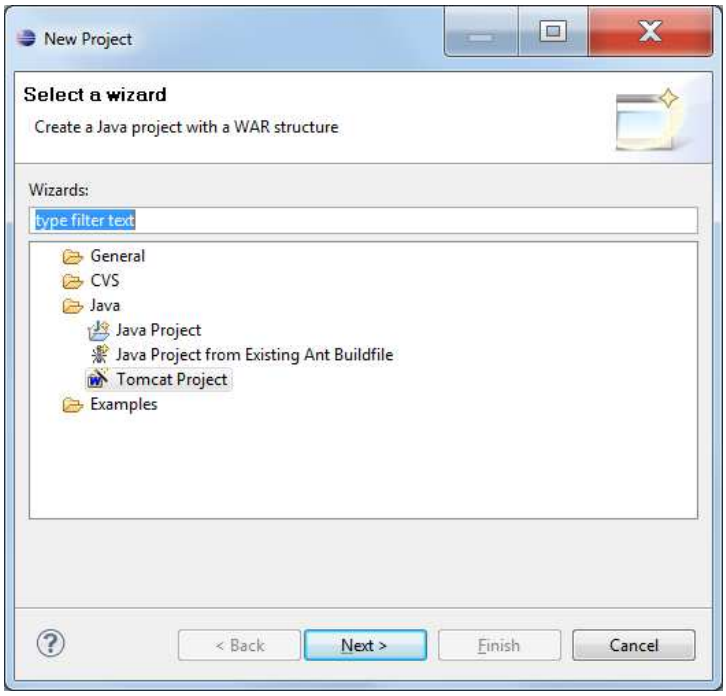


Figura 4.2.12 Asistente para la creación de un proyecto en Eclipse.

Una vez hecho esto habrá que darle nombre al proyecto, después no serán necesarias más configuraciones para empezar a trabajar en el proyecto de desarrollo Web, así que ya se le puede dar click al botón de Finish del asistente de creación de proyectos.

ESTRUCTURA DE ARCHIVOS DEL SISTEMA.

Una vez creado el proyecto de Tomcat Eclipse generará una estructura de archivos para el proyecto por defecto, que a continuación se describirá a detalle, más los directorios que se crearon específicamente para el SIGEB que no vienen incluidos en la configuración inicial de Eclipse.

La estructura de archivos que componen el proyecto es la siguiente:

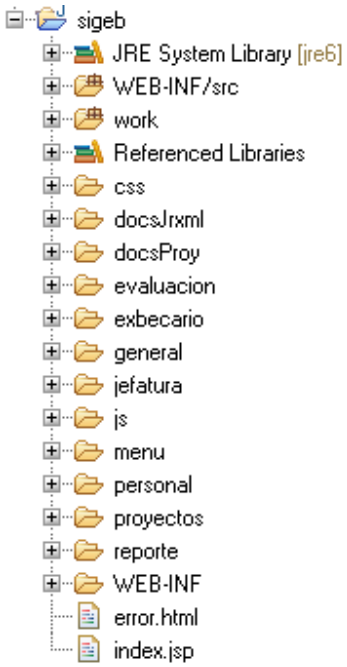


Figura 4.2.13 Vista general de la estructura de archivos que componen el SIGEB.

Se continuará con la explicación de la función de cada uno de los directorios que conforman el SIGEB.

SIGEB raíz.

Directamente en la carpeta raíz del proyecto se encuentran solo 2 archivos, el index.jsp que es la página principal del proyecto, aparecerá siempre que se intente entrar por primera vez al sistema o se haya salido del portal anteriormente, el siguiente archivo error.html funciona cuando el navegador no soporte o se haya deshabilitado la funcionalidad de javascript, esto es debido a que resulta primordial el uso de javascript para el correcto funcionamiento del sistema y esta página incluye ayuda para algunas versiones del navegador soportadas por el SIGEB y las indicaciones para poder habilitar la funcionalidad de javascript.

JRE System Library [jre6].

Como su nombre lo indica en esta sección se ubican las referencias a las paqueterías necesarias para ejecutar cualquier aplicación en Java como el compilador de java y la Máquina Virtual de Java (JVM por sus siglas en inglés). Para fines de aclaración, la ubicación de dichas paqueterías puede variar dependiendo de la máquina en donde se encuentre alojado el sistema, recalcando que el SIGEB es un sistema multiplataforma que puede ser puesto en marcha por diferentes sistemas operativos, solo que como requisito sustancial dicha máquina debe contar con Java instalado así como con el un motor para servlets, en nuestro caso es Tomcat.

Figura 4.2.14 Estructura y ubicación de las librerías básicas de Java.

WEB-INF/src.

En esta carpeta se encuentran alojadas las clases que son creadas por el usuario para que sean utilizadas posteriormente por los archivos jsp que compongan el proyecto, para el caso específico del SIGEB esta carpeta sólo contiene 2 paquetes de clases java: el paquete classes que es el que contiene las clases principales de las que hace uso el SIGEB para su correcto funcionamiento y el paquete conexiones que contiene el ConnectionFactory que nos permite crear conexiones al servidor de base de datos para poder hacer uso del repositorio de información del que se vale el SIGEB. En la sección de documentación del sistema se halla la documentación adecuada de cada una de las clases involucradas.

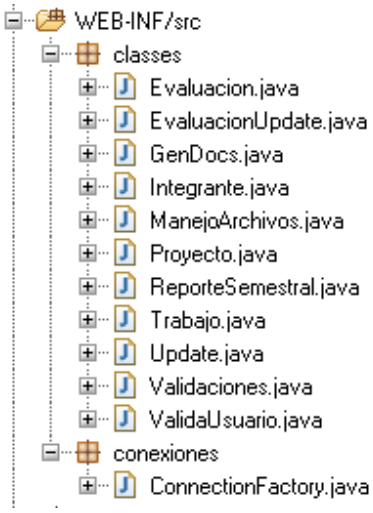


Figura 4.2.15 Estructura y ubicación de las clases creadas por el programador.

Referenced Libraries.

En esta sección se encuentran las referencias a los archivos jar correspondientes a Tomcat, más algunos que considere necesario el desarrollador para poder continuar con el proyecto. Aquí es donde se muestra la configuración del Classpath editada por el programador, como por ejemplo aquí se especifica la ubicación del JDBC para la conexión a bases de datos, también las paqueterías necesarias de las que se vale el SIGEB para la creación de documentos, etc. Resumiendo, la funcionalidad de la sección Referenced Libraries es la de mostrar las referencias a los archivos jar externos a java para que los programas hechos puedan funcionar correctamente.

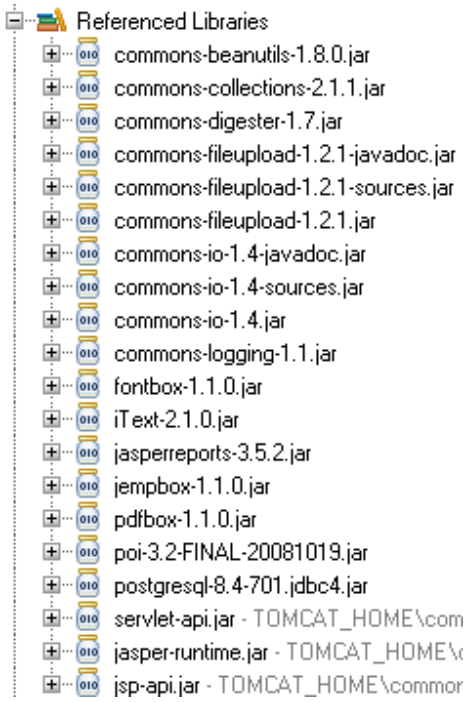


Figura 4.2.16 Archivos jar configurados al classpath por el programador.

Css.

En este directorio se ubican los archivos de hojas de estilo, es decir, los que permiten darle el diseño y la vista al portal Web, también incluye las imágenes que se utilizan en el sistema, para terminar, el propósito de este directorio es muy claro: alojar los archivos que guardan las configuraciones de diseño y vista del SIGEB.

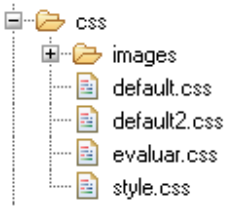


Figura 4.2.17 Archivos de hojas de estilo e imágenes del SIGEB.

DocsJrxml.

Para que el SIGEB pueda generar la documentación de proyectos de forma adecuada, se tiene que valer de archivos de configuración los cuales se encuentran en el directorio docsJrxml, este directorio guarda los archivos .jasper y .jrxml para la generación automática de documentos usando la tecnología JasperReports y iReport.

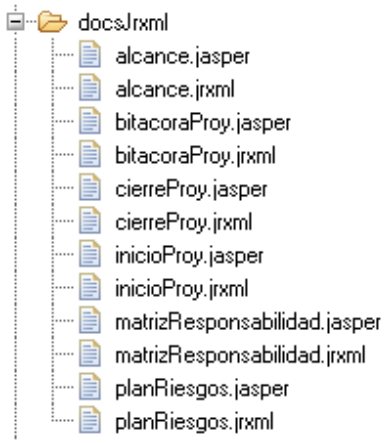


Figura 4.2.18 Archivos de configuración para la generación de documentos.

DocsProy.

Aquí es donde se alojarán todos los documentos relacionados con un proyecto. En el momento en el que se crea un proyecto el sistema creará primero una carpeta con el nombre de las siglas del proyecto que también cambiará de nombre si las siglas del proyecto creado son editadas, una vez creado dicho directorio, el sistema guardará en esa ruta la documentación inicial del proyecto, con opción de que se puedan seguir agregando más archivos relacionados al proyecto, de esta manera se evita que se mezclen los archivos de diferentes proyectos y facilita la depuración de archivos en servidor.

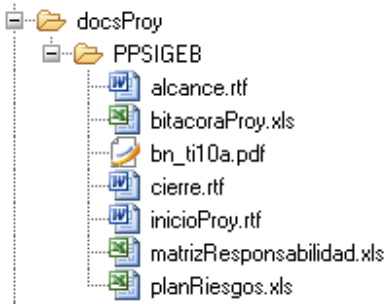


Figura 4.2.19 Ejemplificación de la estructura de archivos de la documentación de proyectos.

Evaluación.

Esta carpeta contiene los archivos correspondientes al módulo de evaluación de becarios, toda la funcionalidad de dicho módulo se encuentra debidamente separada en esta carpeta donde la función de cada archivo se explicará en los apartados correspondientes.

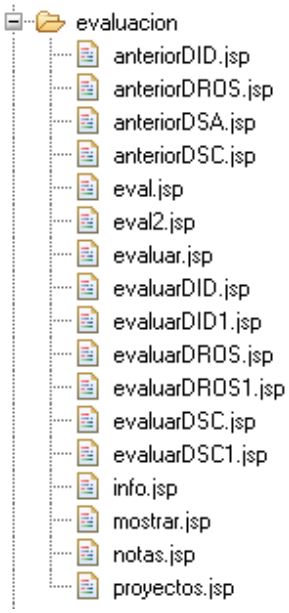


Figura 4.2.20 Archivos correspondientes al módulo de evaluación.

Exbecario.

Esta carpeta contiene los archivos correspondientes al módulo de exbecarios, toda la funcionalidad de dicho módulo se encuentra debidamente separada en esta carpeta donde la función de cada archivo se explicará en los apartados correspondientes.

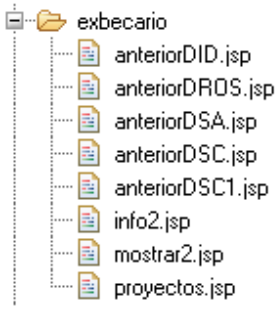


Figura 4.2.21 Archivos correspondientes al módulo de exbecarios.

General.

Esta carpeta contiene los archivos generales del sistema, es decir, las funcionalidades independientes del usuario que haya ingresado al sistema como por ejemplo el cambio de contraseña, la búsqueda de algún becario o el calendario con la fecha actual.

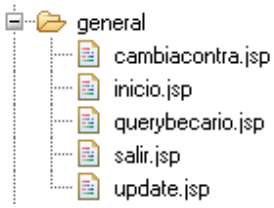


Figura 4.2.22 Archivos correspondientes a las generalidades del sistema.

Jefatura.

Esta carpeta contiene los archivos correspondientes al jefe de la unidad, en esta sección se comprende una versión simplificada de todo el sistema en el cual sólo se han habilitado las vistas de lo realizado por cada jefe del departamento, es decir, las evaluaciones hechas, los proyectos registrados, etc., sin opción a poder editar algo que haya sido hecho.

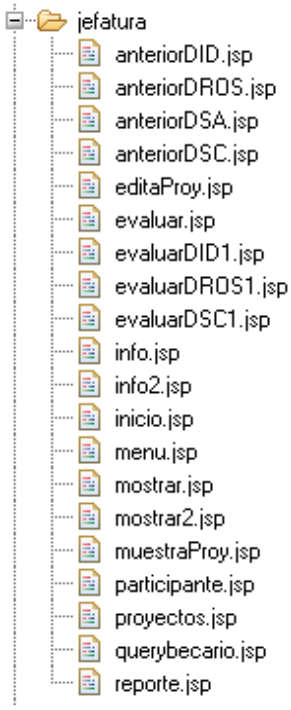


Figura 4.2.23 Archivos correspondientes al jefe de la unidad.

Js.

Directorio que aloja los códigos fuente de las tecnologías javascript y AJAX, todo lo que hace el SIGEB de forma asíncrona se encuentra definido en estos archivos de extensión js, primordiales para el correcto funcionamiento del sistema.

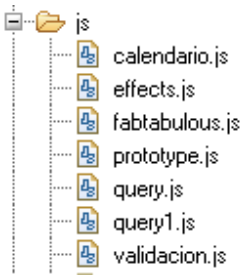


Figura 4.2.24 Archivos correspondientes a los códigos fuente de javascript y AJAX.

Menú.

Directorio que contiene los archivos correspondientes al menú principal del sistema, contienen las funcionalidades de mandar a llamar a todos los demás módulos, además de unificarlos en el archivo de pantalla principal del sistema, todo esto obedeciendo al uso de la programación modular de sistemas Web, estos archivos son pieza clave en el funcionamiento del SIGEB.

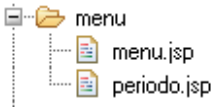


Figura 4.2.25 Archivos correspondientes al menú principal del sistema.

Personal.

Directorio que almacena las fotografías de todos los becarios de la Unidad ordenadas por el RFC de cada becario. Debido a la gran cantidad de imágenes que abarca, se trasladaron a una carpeta aparte de las imágenes del sistema.

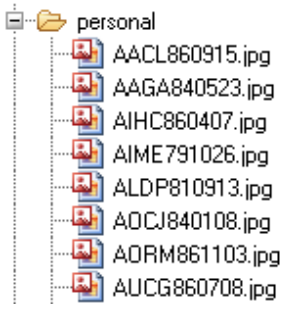


Figura 4.2.26 Fotografías de los becarios de la Unidad.

Proyectos.

Esta carpeta almacena los archivos correspondientes al módulo de proyectos, toda la funcionalidad de dicho módulo se encuentra debidamente separada en esta carpeta donde la función de cada archivo se explicará en los apartados correspondientes.

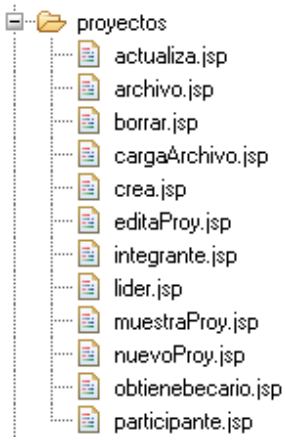


Figura 4.2.27 Archivos correspondientes al módulo de proyectos.

Reporte.

Esta carpeta aloja el archivo correspondiente al módulo de reporte semestral, toda la funcionalidad de dicho módulo se encuentra debidamente separada en esta carpeta donde la función de cada archivo se explicará en los apartados correspondientes.

Figura 4.2.28 Archivo correspondiente al módulo de reporte semestral.

WEB-INF/lib.

Ubicación física de los archivos jar necesarios para el correcto funcionamiento del SIGEB, es necesario incluir dichos archivos para mantener la propiedad del SIGEB de ser multiplataforma, así el servidor en el que se instale contará con los archivos físicos que necesita el SIGEB.

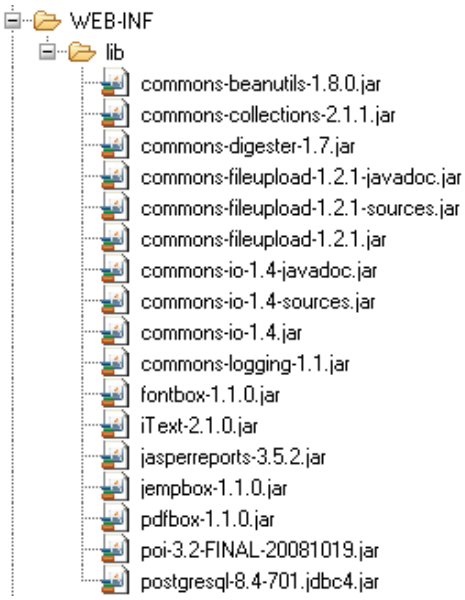


Figura 4.2.29 Ubicación física de las paqueterías que utiliza el SIGEB.

La arborescencia del proyecto.

En una sola página se busca definir la funcionalidad del proyecto así como la navegación que tendrá que hacer el usuario a través de los archivos.

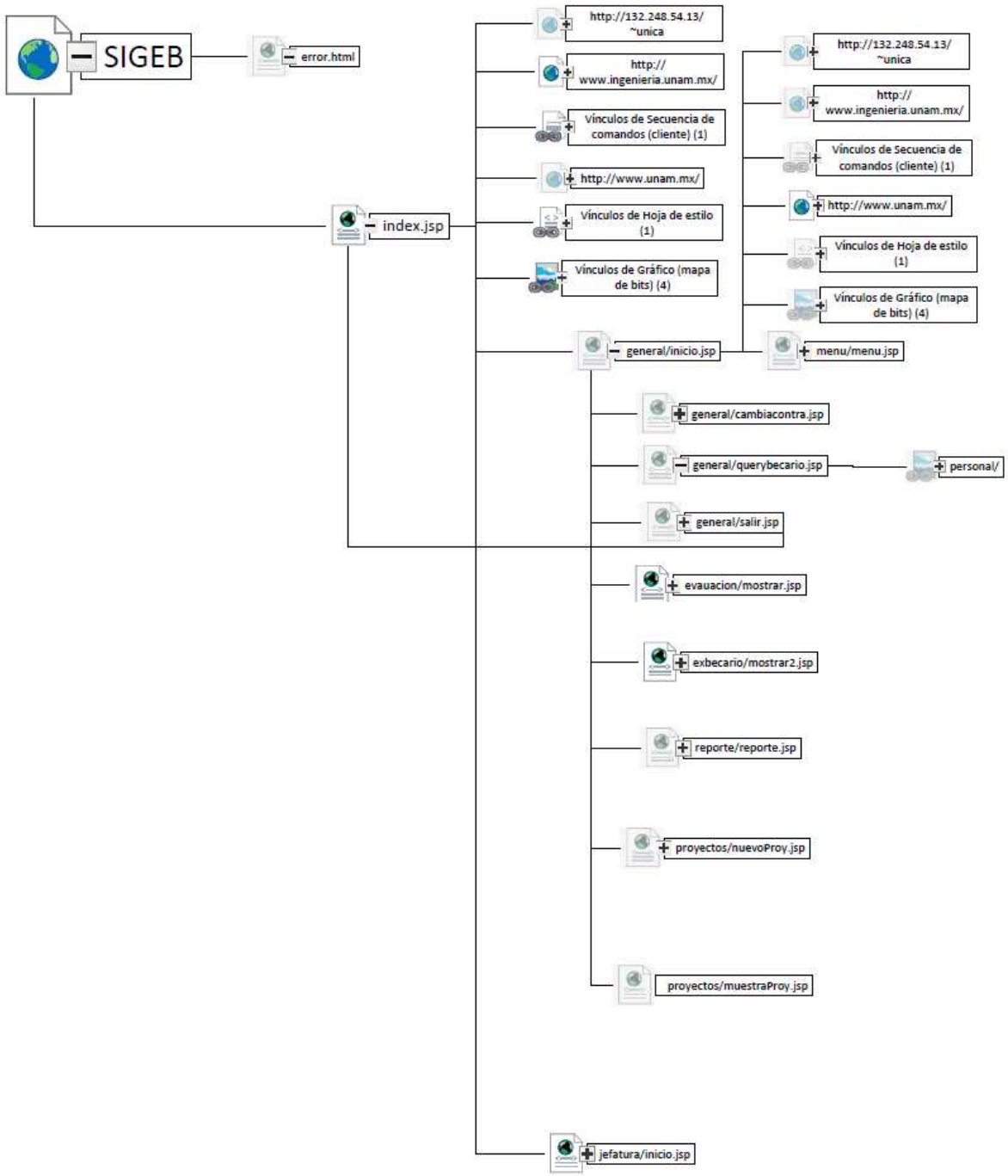


Figura 4.2.30. Arborescencia principal del SIGEB.

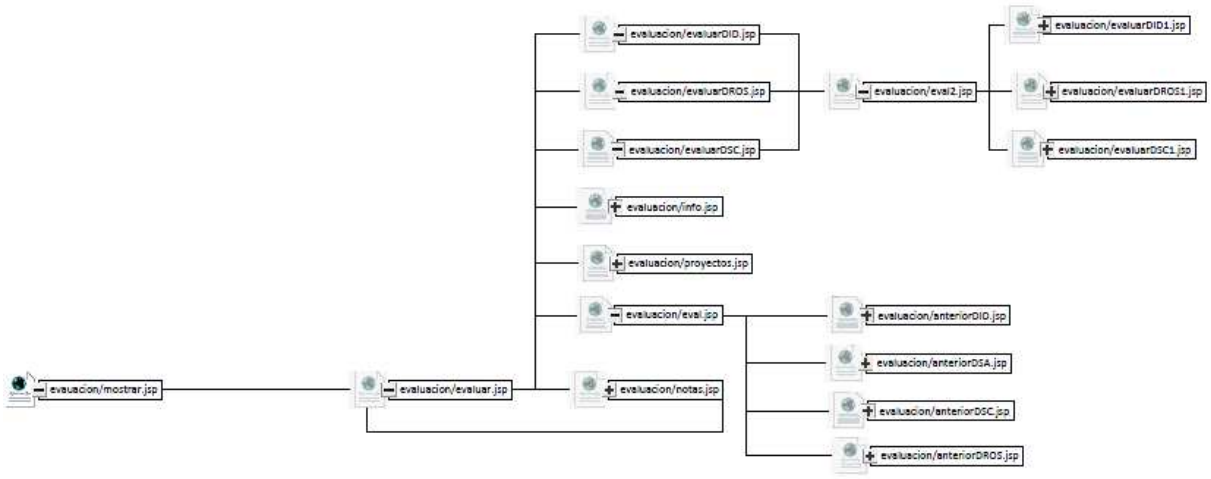


Figura 4.2.31. Arborescencia del paquete evaluación.

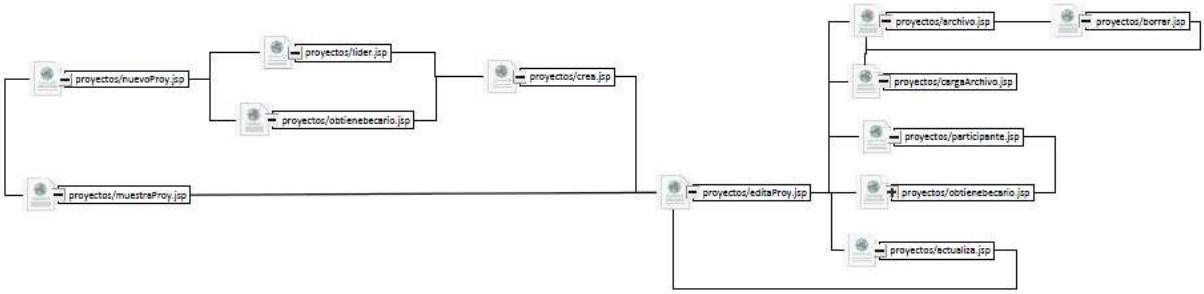


Figura 4.2.32. Arborescencia del paquete proyectos.

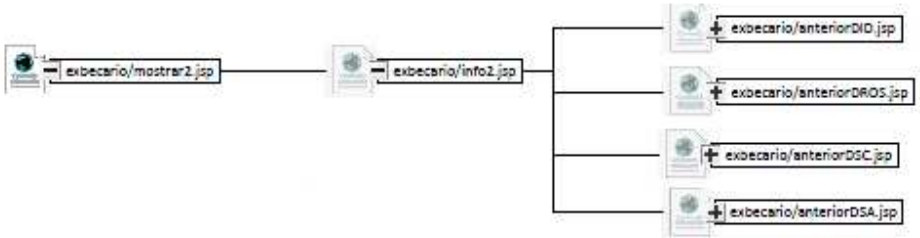


Figura 4.2.33. Arborescencia del paquete exbecario.

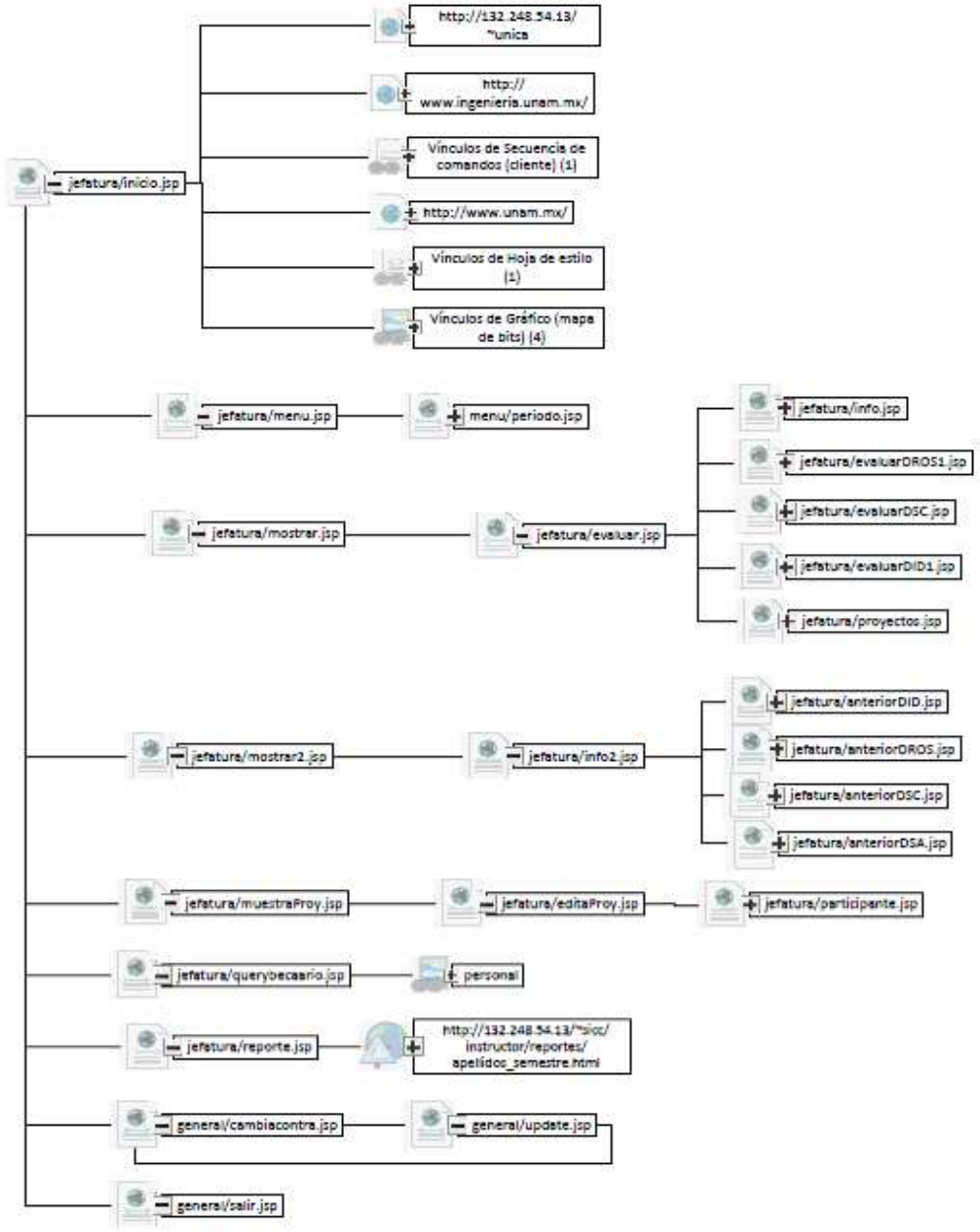


Figura 4.2.34. Arborescencia del paquete jefatura.

4.3 Programación segura de páginas Web.

Acerca del software seguro para páginas web.

Para el campo de las aplicaciones de Internet y la cantidad de veces que se han reportado vulnerabilidades que permite acceder a las bases de datos de los clientes por culpa de fallos en los desarrollos, la seguridad juega un papel muy importante en estos productos y no se le presta la atención que merece. En cuyos casos donde los desarrolladores reportan la ausencia de requisitos de seguridad a sus responsables y las excusas por parte del equipo de diseño siempre suelen ser las mismas: no hay tiempo, los plazos son escasos, no hay presupuesto. Pero luego a quienes se apunta con el dedo cuando empiezan a aparecer los bugs, es a los desarrolladores.

Las organizaciones deben definir una serie de prioridades y la mayoría de las veces la seguridad no suele ser de alta prioridad. A menudo, no es rentable hacer un sistema tan seguro como sea posible, ya que el riesgo es bajo y el coste es alto. El software está en la raíz de la mayoría de los problemas de seguridad informática, si éste no se comporta de forma adecuada surgirán problemas de integridad, disponibilidad, confidencialidad... Los bugs y vulnerabilidades son la causa de un mal diseño y una mala implementación. **La seguridad debe estar presente en todas las fases del ciclo de vida de un producto**.

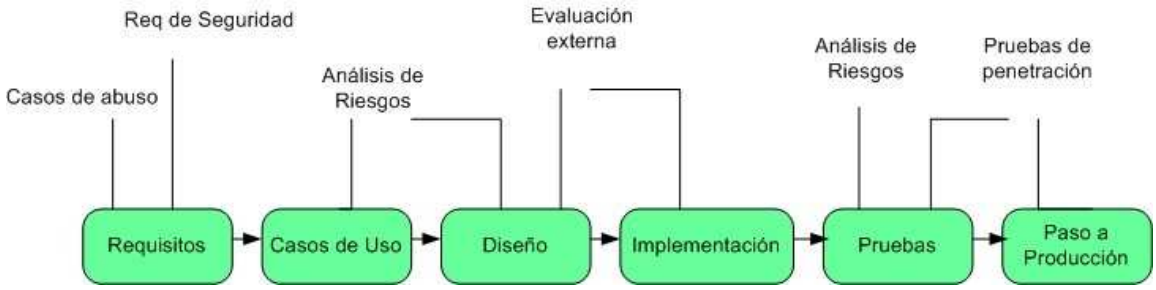


Figura 4.3.1 Inclusión de la seguridad en el ciclo de vida del software.

En el inicio, durante la fase de análisis se debe realizar un análisis de impacto y pensar en las amenazas y vulnerabilidades a las que puede verse afectado el sistema, para ello se deben incorporar modelos de amenazas. Además todo arquitecto debe recoger: las políticas, los riesgos y las normas jurídicas.

En la fase de diseño se debe realizar un análisis de riesgos, el cual debe ser realizado por expertos en seguridad quienes son capaces de reconocer en que situaciones se producen los ataques más comunes. Durante la fase de implementación, si los programadores necesitan formación en seguridad, se les debe de proporcionar dicha formación.

En el plan de pruebas funcionales, también debe estar presente la seguridad ya que pueden surgir nuevas salvaguardas a incluir. Se deben incluir pruebas de seguridad, test de intrusión, pruebas de carga, pruebas de denegación de servicio y planes de mitigación de riesgos.

PROGRAMACIÓN WEB MODULAR.

El tener una web modular quiere decir que no todo el sistema debe de estar presente en un solo archivo, en el caso del SIGEB un archivo .jsp. Habrá que dividir dicho sistema en diferentes archivos de código fuente que se repartirán las funciones del sistema para que sea más fácil tanto el mantenimiento del sistema así como la prevención de ataques al mismo que es lo que se tratará en este tema.

En el siguiente ejemplo de una página del SIGEB se podrá observar la inclusión de diferentes archivos que reparten parte de las funciones.

El archivo inicio.jsp dentro de la carpeta general será nuestro objeto de estudio.

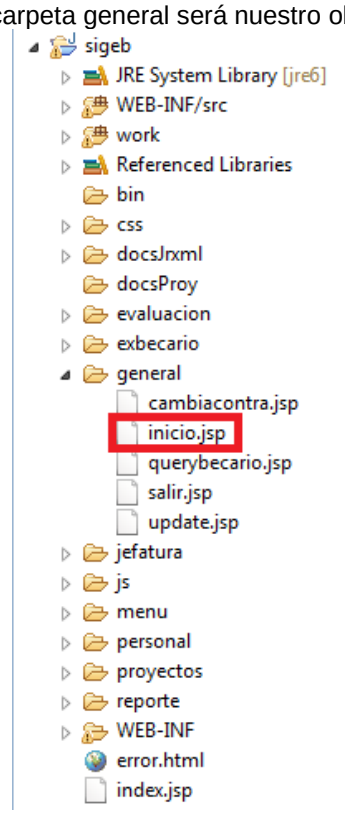


Figura 4.3.2 Ubicación del archivo de inicio.jsp.

Revisando el código podremos ver que el código correspondiente al menú se manda a llamar con una directiva de inclusión de la siguiente manera:

```
<%/*COLOCAR EL ARCHIVO DEL MENU*/%>
        <%@ include file="/menu/menu.jsp" %>
```

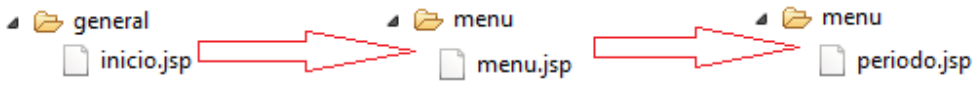
Ahora revisando el archive de menú podremos observar que ahí está el código correspondiente al diseño y a la funcionalidad del menú principal del SIGEB.

```
<ul>
    <li>
    <h2>Men&uacute;</h2>
    <ul>
        <li><form id="bec" name="bec" method="post" action="">
        <font color="#000066">Periodo</font>
        <jsp:include page="../menu/periodo.jsp"></jsp:include>
        </form></li>
        <li><a href="javascript: showEval('<%=siceb.getAttribute("depto")%>');"
        class="Login" tabindex="3">Evaluaci&oacute;n</a>
        </li>
        <li><a href="javascript: showExbecario('<%=siceb.getAttribute("depto")%>');"
        class="Login" tabindex="3">Ex Becarios </a></li>
```

```
                <li><a                                                    href="javascript:
                showReporte('<%=siceb.getAttribute("depto")%>',document.bec.anyo.value);"
                class="Login" tabindex="3">Reporte Semestral </a></li>
        </ul>
        </li>
        <li>
        <h2>Proyectos</h2>
        <ul>
                <li><a          href="javascript:       showProy('<%=siceb.getAttribute("depto")%>');"
                class="Login" tabindex="3">Proyectos</a></li>
                <li><a          href="javascript:nuevoProy(document.bec.anyo.value,          true,
                '<%=siceb.getAttribute("depto")%>')" class="Login">Nuevo proyecto </a></li>
        </ul>
        </li>
</ul>
```

Que a su vez también hace un llamado al archivo de periodo que hace que siempre se muestre el último periodo por defecto.

Finalmente quedarían los llamados a archivos de la siguiente forma gráfica:



Y como resultado final, el menú de nuestra página de inicio donde el usuario final no puede apreciar que se trató de varios archivos incluidos en uno solo.

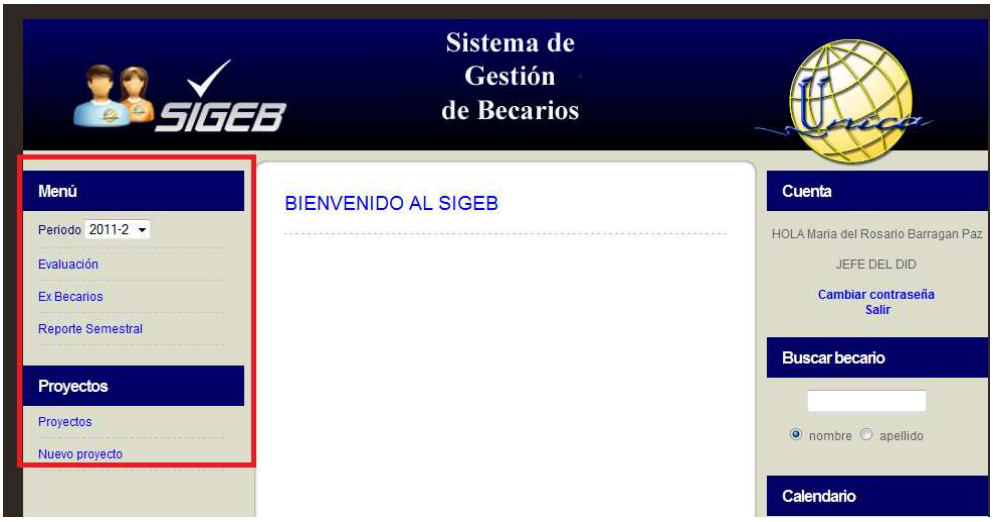


Figura 4.3.3 Módulo del menú funcionando en el sistema.

CONEXIONES SEGURAS.

Una mala práctica de programación consiste en revelar ubicaciones de servidor, así como datos de usuario y contraseñas en los códigos, un ejemplo de esto está en la clase ConnectionFactory que permite crear conexiones al servidor de base de datos y es donde se incluye en la url la dirección ip del servidor de base de datos, el Software Manejador de Base de Datos, la base de datos a la que se va a conectar, el usuario que se va a conectar y la contraseña.

Una manera de evitar esto que además elimina la necesidad de modificar el código para poder cambiar de servidor de base de datos y así no tener que recompilar el código del ConnectionFactory es mediante un archivo .properties que contendrá los datos de conexión antes mencionados de la siguiente manera.

```
#Nombre del driver para postgreSQL
connection.database.driver.class = org.postgresql.Driver
#URL de la conexion a la base de datos del SIGEB
connection.database.url = jdbc:postgresql://127.0.0.1/SICEBV2?user=postgres&password=admin
#URL de la conexion a la base de datos del SICC
connection.database.url1 = jdbc:postgresql://127.0.0.1/BDSICC?user=postgres&password=admin
```

La estructura es la siguiente: El nombre de la propiedad del lado izquierdo, el signo de igual y en el lado derecho de la igualdad los datos que contendrá dicha propiedad, de esta forma se puede mejorar tanto la administración como el control de los datos de una conexión en un solo archivo de texto plano.

La forma de hacer uso de las propiedades de dicho archivo en la clase ConnectionFactory es la siguiente:

```
import java.sql.*;
import java.util.*;
import java.io.*;

public class ConnectionFactory {
	private static final Properties properties = new Properties();
	//Cargar el driver
	private static final ConnectionFactory cf = new ConnectionFactory();
	//Constructor privado
	private ConnectionFactory(){
		String driverClass;
		try{
			String path = “Dirección absoluta del archivo .properties";
			properties.load(new FileReader(new File(path)));
			driverClass = properties.getProperty("connection.database.driver.class");
			Class.forName(driverClass);
		}catch(FileNotFoundException e){
			throw new RuntimeException("Archivo de configuracion no encontrado", e);
		}catch(IOException e){
			throw new RuntimeException("Error al leer el archivo de configuracion", e);
		}catch(ClassNotFoundException e){
			throw new RuntimeException("Driver no encontrado, revise el classpath",
e);
		}
	}
	//Obtener la conexion a la BD
	public static Connection getConnection(){
		String url = properties.getProperty("connection.database.url");
		try{
			return DriverManager.getConnection(url);
		}catch (SQLException e){
			throw new RuntimeException("Error al obtener la conexion a la base de
datos", e);
		}
	}
	public static Connection getConnectionSicc(){
```

```
                String url = properties.getProperty("connection.database.url1");
                try{
                        return DriverManager.getConnection(url);
                }catch (SQLException e){
                        throw new RuntimeException("Error al obtener la conexion a la base de
datos", e);
                }
        }
        //Cerrar la conexion
        public static void close(Connection connection){
                try{
                        if(connection != null){connection.close();}
                }catch(SQLException e){
                        System.out.println("Error al cerrar la conexion");
                        e.printStackTrace();
                }
        }
        //Cerrar un statement
        public static void close(Statement statement){
                try{
                        if(statement != null){statement.close();}
                }catch(SQLException e){
                        System.out.println("Error al cerrar la conexion");
                        e.printStackTrace();
                }
        }
        //Cerrar el ResultSet
        public static void close(ResultSet rs){
                try{
                        if(rs != null){rs.close();}
                }catch(SQLException e){
                        System.out.println("Error al cerrar la conexion");
                        e.printStackTrace();
                }
        }
}
```

Finalmente como acotación se puede observar que la clase ConnectionFactory respeta el patrón singleton para los proyectos Web que consiste en crear una sola instancia de conexión (por ello el constructor privado) para todo el proyecto y así disminuir la carga al servidor de base de datos.