

# **WEB APP**



# **JAVA EE**

## Indice

Introducción:.....	3
Presupuesto:.....	6
Manual de usuario:.....	7
Documentación técnica:.....	10
Diagrama E-R.....	10
MVC.....	11
Estructura.....	13
JSP.....	14
Servlets:.....	14
Dispatcher.....	15
Enterprise Java Beans(EJB).....	16
Session beans.....	17
Stateless.....	18
Stateful.....	19
Singleton.....	19
Patrones de programación.....	20
Interface.....	20
Facade.....	21
Singleton.....	23
Arraylist y HashMap:.....	24
Entity manager:.....	28
Referencias:.....	31
Mejoras de la aplicación:.....	32
Conclusión:.....	32
Carta de recomendación:.....	33

## Introducción

La motivación del proyecto es plasmar en una aplicación informática la idea de negocio que nos llevó a ganar el concurso de la fundación Repsol para la provincia de la Coruña con el proyecto Crowdsolving y así tener algo sólido y real que mostrar a los potenciales inversores.

<https://www.corunahoy.es/o-proxecto-CROWD-SOLVING-gana-a-primeira>  
[https://www.lavozdeg Galicia.es/noticia/educacion/2018/03/15/alumnos-fp-unen-empresas-estudiantes-resolver-retos/0003\\_201803H15C6991.htm](https://www.lavozdeg Galicia.es/noticia/educacion/2018/03/15/alumnos-fp-unen-empresas-estudiantes-resolver-retos/0003_201803H15C6991.htm)

Crowdsolving es una plataforma tecnológica que permite una interacción directa entre centros de estudios y compañías, de tal forma que los estudiantes tengan acceso a proyectos/retos planteados por las empresas acorde a las necesidades de las mismas.

El objetivo técnico es conseguir una aplicación web con base de datos con las siguientes funcionalidades:

Añadir ,Borrar y Mostrar los registros de 3 Entidades/Clases:

Empresa: es la encargada de proponer el Reto a resolver

Reto: es el problema que quiere solucionar la empresa, es resuelto por un equipo.

Equipo: es el conjunto de personas de distintas disciplinas encargado de resolver el reto.

La aplicación permite que los usuarios y las empresas se puedan registrar en la plataforma , guardando dichos registros en una base de datos.

También permite ver en el navegador los cambios en tiempo real sin necesidad de utilizar programas externos como MySQL workbench y PHPmyadmin.

Por otra parte me sirve para afianzar conocimientos en una tecnología tan utilizada como java EE y con gran demanda profesional.

Por ejemplo el grupo Inditex y todas las auxiliares trabajan con Java EE. Se elige la tecnología JAVA E.E. por ser la tecnología cursada en Desarrollo de aplicaciones en servidor, además de la tecnología utilizada en la empresa de prácticas .

La principal razón por la que se ha elegido Java es su escalabilidad, en este sencillo ejemplo tenemos 3 relaciones .Pero utilizando la Abstract facade que se explica en la descripción técnica podríamos manejar 10 o más relaciones con una estructura bastante similar , con las mismas interfaces y métodos.

Otro punto fuerte es su administración de memoria automatizada, que consigue una aplicación web eficiente.

Java EE es una de las tecnologías más demandadas por las empresas, he estado realizando entrevistas para empresas en Sofía :

Concretamente Vivacom y Sitel.(dos empresas telefónicas)

Además permite la portabilidad a Android(está basado en Java) en caso de querer realizar la versión para móvil.

Todas eran para programador junior y en todas pedían Java EE.

Por lo que he visto una gran oportunidad en este proyecto para refrescar y mejorar mis conocimientos en Java. Las dos entrevistas tenían una parte técnica donde te pedían programar unos métodos delante de ellos para comprobar tus habilidades sobre el terreno.

*Por lo tanto considero que toda práctica implementando código es poca.*

## **Tecnologías y herramientas utilizadas:**

**Tomcat : se utiliza con Glassfish porque es necesario para recorrer los objetos con Jstl**

Glassfish: maneja un amplio espectro de los componentes de JavaEE entre ellos los beans.

Sirve para desplegar la aplicación

Hibernate: es un framework que sirve para realizar las conexiones entre modelo y base de datos. Requiere realizar correctamente las anotaciones correspondientes

Html:

Lenguaje de marcas que Sirve para visualizar el programa por parte del usuario

XML :

Lenguaje de marcas que sirve como standard para intercambiar información de forma estructurada por ejemplo una base de datos.

Netbeans :Es el IDE elegido compila el código y es necesario para ejecutar la aplicación.Es imprescindible utilizar un IDE para programar en Java .

## Presupuesto:

Para el presupuesto se pone el precio de programador a 22eur/h (iva incluido), para el despliegue utilizaremos un programador senior (35eur/h) en la siguiente tabla se desglosan los gastos del presupuesto.

No tenemos partida de software porque todos los utilizados son libres(java,mysql,css,html).

Esta parte incluye las mejoras descritas en el último apartado(creo que tiene mucho sentido porque incrementan la calidad del producto final a un coste razonable)

Lo he programado con mi pc por tanto no he incluido coste de hardware.

Los gastos se distribuyen de la siguiente manera.

Dominio	12 eur
Hosting	50 eur
Horas programadas(50h)	1.100 eur
Despliegue aplicación profesional(3h)	105 eur
Mejoras visuales(10h)	220 eur
Total	1.487 eur

Se añade un 10% adicional por si ocurre algún imprevisto del estilo del despliegue de la aplicación.

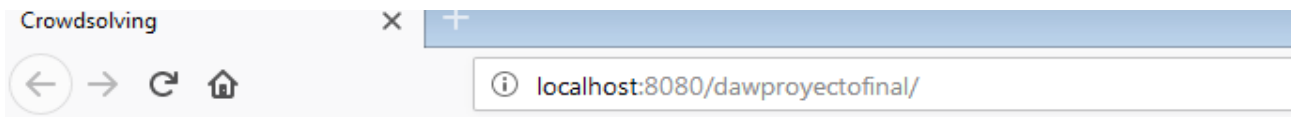
Por tanto el presupuesto final es de 1.635,7 euros.

## Manual de usuario:

La idea es que el usuario final acceda a ella a través del navegador con la aplicación ya desplegada.

A continuación se presentan las instrucciones necesarias para introducir los datos e interactuar con los menús.

1) se abre el menú principal en el navegador que consta de 2 posibilidades añadir que sirve para agregar un nuevo registro(Empresa,Reto o equipo) a la base de datos y mostrar que despliega como resultado los registros existentes



### Empresas

- [Añadir](#)
- [Mostrar](#)

### Retos

- [Añadir](#)
- [Mostrar](#)

### Equipos

- [Añadir](#)
- [Mostrar](#)

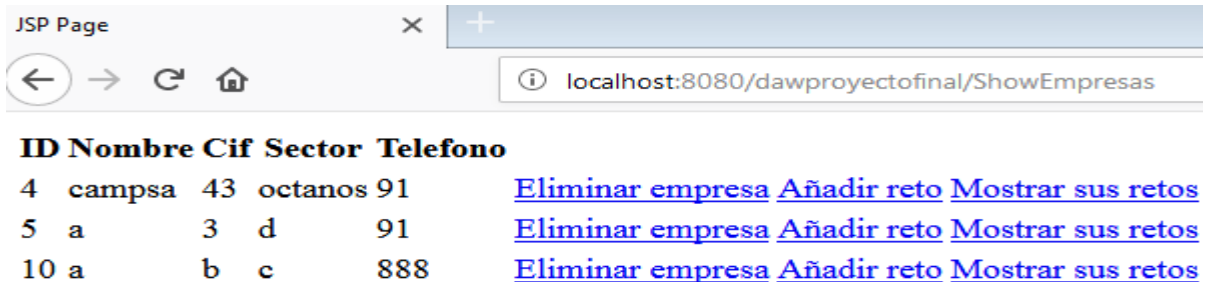
en la base de datos y despliega un submenú.

2) tras pulsar mostrar , se muestran las empresas existentes en la base de datos.

Muestra las siguientes opciones

a)asociar reto a empresa b)eliminar la empresa c)mostrar los retos asociados a la empresa en la base de datos

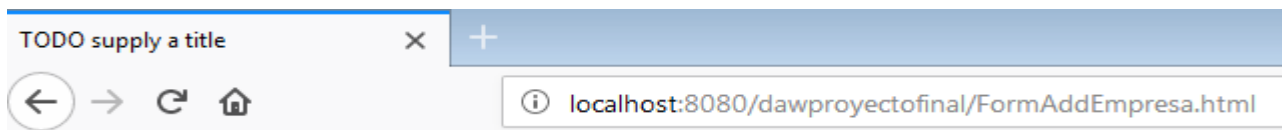
3)  
tras



The screenshot shows a web browser window with the address bar displaying 'localhost:8080/dawproyectorfinal/ShowEmpresas'. The page content is a table with the following data:

ID	Nombre	Cif	Sector	Telefono	
4	campsa	43	octanos	91	<a href="#">Eliminar empresa</a> <a href="#">Añadir reto</a> <a href="#">Mostrar sus retos</a>
5	a	3	d	91	<a href="#">Eliminar empresa</a> <a href="#">Añadir reto</a> <a href="#">Mostrar sus retos</a>
10	a	b	c	888	<a href="#">Eliminar empresa</a> <a href="#">Añadir reto</a> <a href="#">Mostrar sus retos</a>

pulsar añadir se despliega el siguiente formulario(control errores html5 y java)



The screenshot shows a web browser window with the address bar displaying 'localhost:8080/dawproyectorfinal/FormAddEmpresa.html'. The page title is 'TODO supply a title'.

## Introduzca nueva empresa

**Nombre**

**Cif**

**Sector**

**Telefono**



4) si lo has hecho correctamente



**La empresa ha sido añadido al registro**

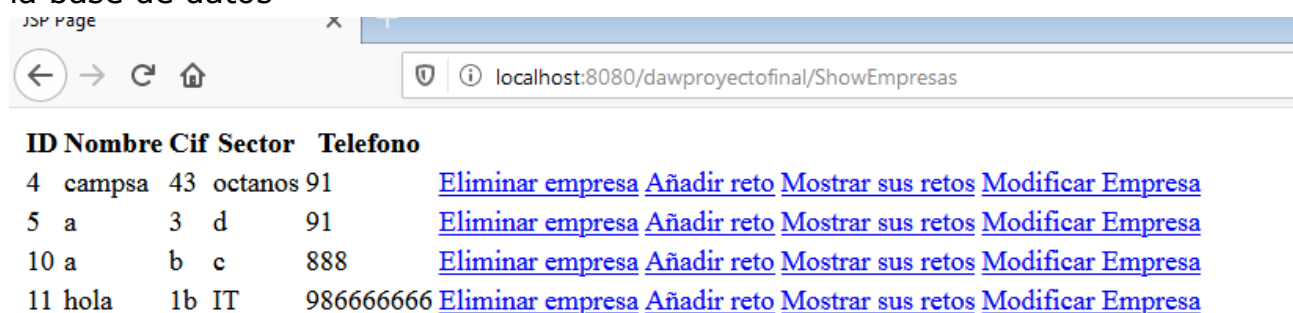
Nombre: hola

cif: 1b

sector: IT

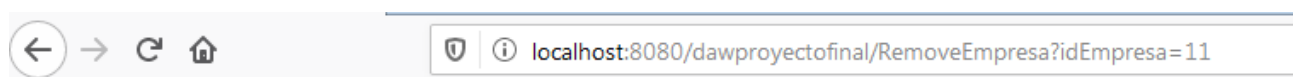
Telefono: 986666666

5) puedes comprobarlo volviendo atras y volviendo a mostrar las empresas de la base de datos



ID	Nombre	Cif	Sector	Telefono	
4	campsa	43	octanos	91	<a href="#">Eliminar empresa</a> <a href="#">Añadir reto</a> <a href="#">Mostrar sus retos</a> <a href="#">Modificar Empresa</a>
5	a	3	d	91	<a href="#">Eliminar empresa</a> <a href="#">Añadir reto</a> <a href="#">Mostrar sus retos</a> <a href="#">Modificar Empresa</a>
10	a	b	c	888	<a href="#">Eliminar empresa</a> <a href="#">Añadir reto</a> <a href="#">Mostrar sus retos</a> <a href="#">Modificar Empresa</a>
11	hola	1b	IT	986666666	<a href="#">Eliminar empresa</a> <a href="#">Añadir reto</a> <a href="#">Mostrar sus retos</a> <a href="#">Modificar Empresa</a>

6) si pulsas eliminar



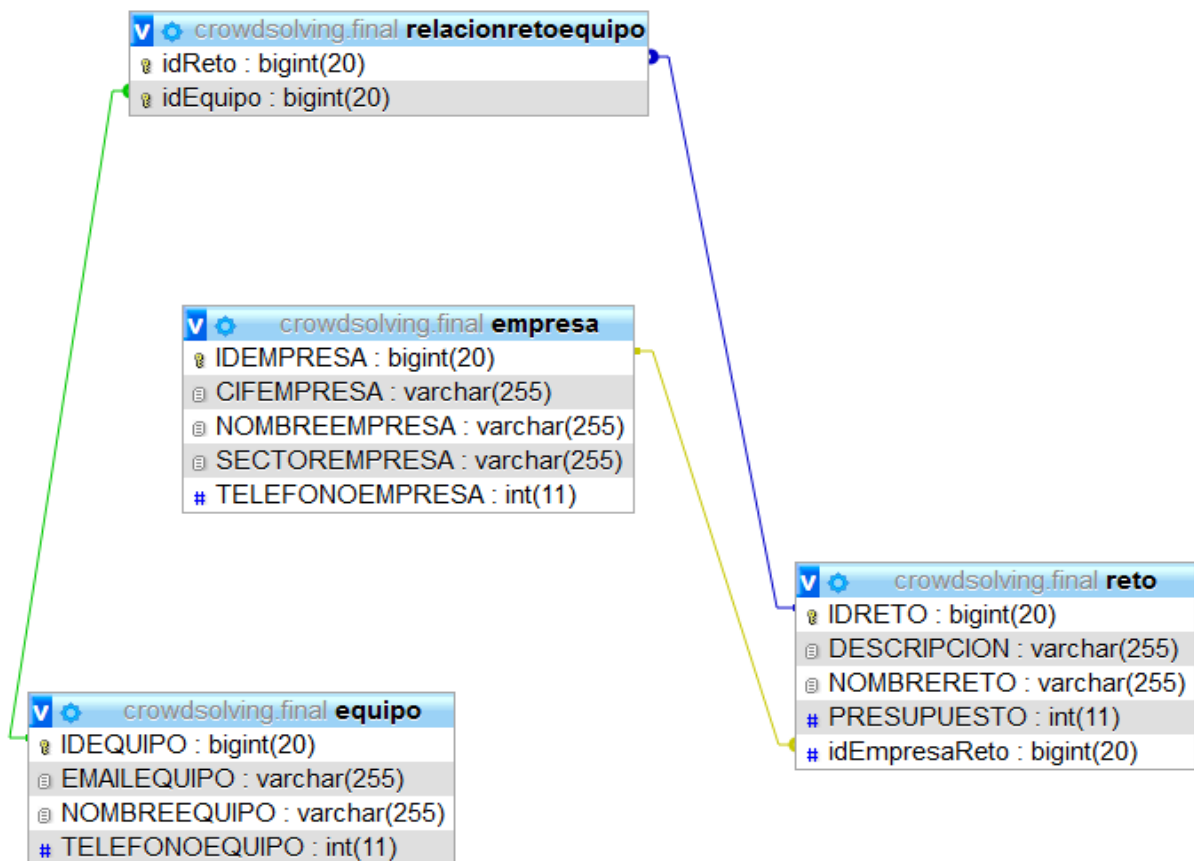
localhost:8080/dawproyectofinal/RemoveEmpresa?idEmpresa=11

**la empresa con identificador 11 ha sido eliminado**

los menús son idénticos para Reto y equipo

# Documentación técnica:

## Diagrama E-R



viendo el diagrama entidad relación de las clases podemos observar que tenemos una relación 1:N entre empresa y reto donde se inserta la clave foránea idEmpresaReto para relacionarlas.

Y una relación M:N entre equipo y reto que requiere de la construcción de una nueva tabla(con las claves primarias de equipo y reto para relacionarlas).

Esta relación también es necesario plasmarla al diseñar las clases en JAVA.

```
private Integer telefonoEmpresa;  
@OneToMany(cascade = CascadeType.ALL, fetch = FetchType.LAZY) // especifica la relacion entre las clases empresa y reto  
  
@JoinColumn(name = "idEmpresaReto", referencedColumnName = "idEmpresa") // hay que especificar las columnas por donde se unen las entidades (su id)  
private List<Reto> retosEmpresa;
```

En la clase Empresa se ha de indicar la relacion @OneToMany y la columna de unión(clave foránea)

```

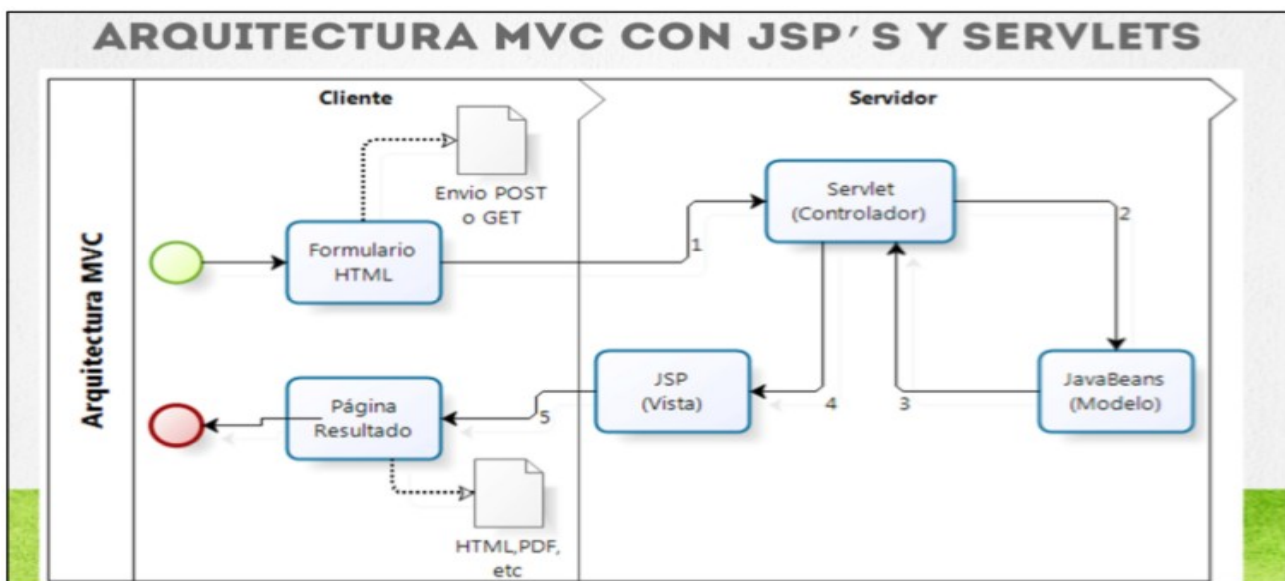
@ManyToMany(cascade = CascadeType.PERSIST, fetch = FetchType.EAGER)
@JoinTable(
    name = "relacionRetoEquipo",
    joinColumns = @JoinColumn(name = "idReto", referencedColumnName = "idReto"), // se especifica la relacion M a n en los 2 sentidos
    inverseJoinColumns = @JoinColumn(name = "idEquipo", referencedColumnName = "idEquipo")
)

```

En la clase reto también se especifica la relación ManyToMany y la tabla por donde se une JoinTable.

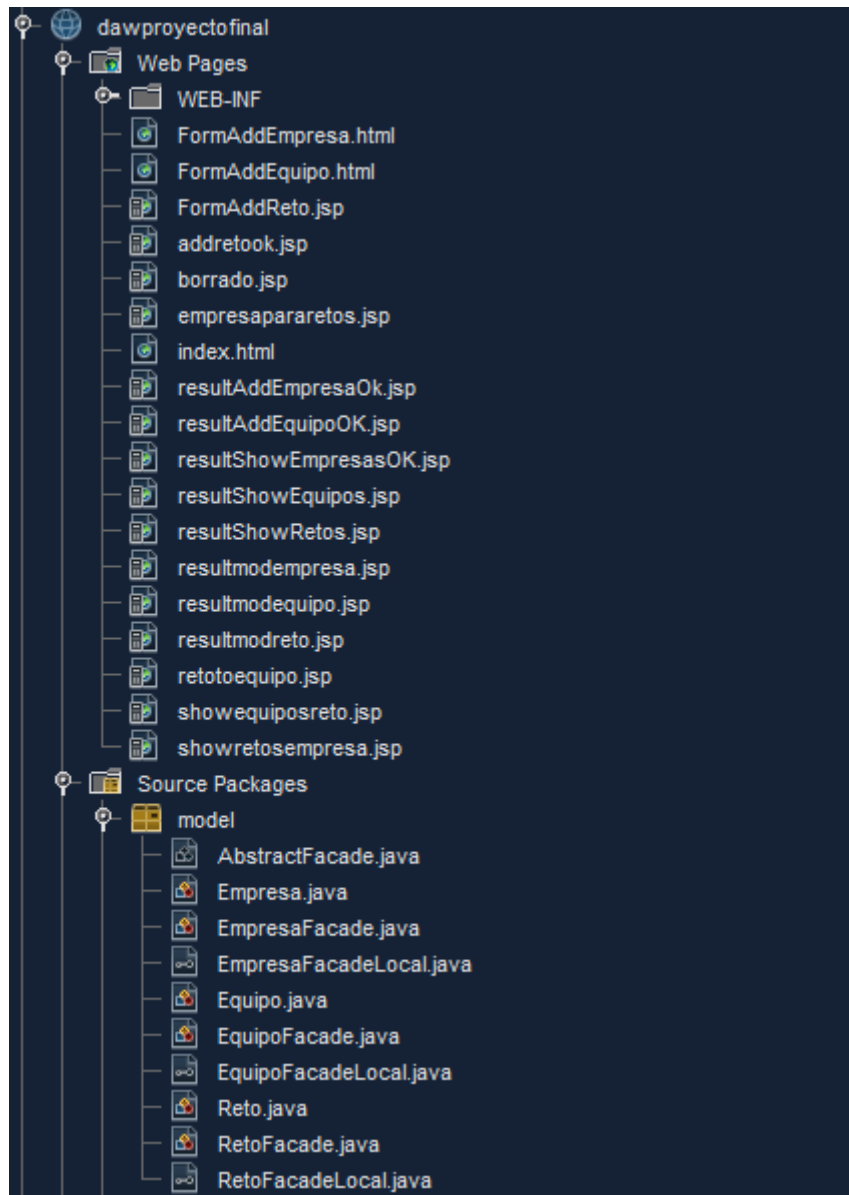
## MVC

La aplicación esta codificada en java utilizando el modelo vista controlador(MVC).En la imagen se muestra el esquema



El modelo vista controlador es un patrón de arquitectura que separa la parte lógica de la aplicación de su representación.

como se puede apreciar en la imagen esta la aplicación esta construída utilizando el patrón MVC:



Como se puede apreciar en la imagen superior:

la vista: esta compuesta de la parte web(la que se muestra al usuario o frontend).

Los archivos que la componen son los correspondientes al diseño en java E.E(jsp)y los más utilizados en diseño web html css.

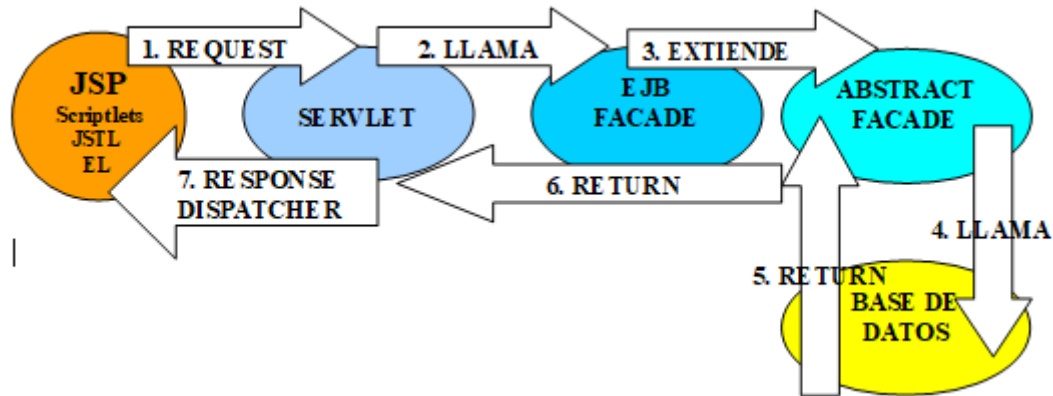
En el modelo :

se encuentra la parte que contiene la lógica (clases y métodos) del programa.**Se puede apreciar la diferencia de complejidad entre el del esquema y el de mi aplicación,para empezar utiliza 3 beans.**

El controlador:lo componen fundamentalmente los Servlets que transmiten instrucciones y datos bidireccionalmente entre la vista y el modelo.

## Estructura

La estructura de la aplicación se resume en el siguiente esquema



aprovecharemos para definir los elementos del esquema:

## JSP

Java Server Pages (JSP) es una tecnología para el desarrollo de páginas web que soporta contenido dinámico. Esto ayuda a los desarrolladores a insertar código Java en páginas HTML utilizando etiquetas especiales JSP, la mayoría de las cuales comienzan con `<%` y terminan con `%>`.

Un componente JavaServer Pages es un tipo de servlet Java que está diseñado para cumplir la función de interfaz de usuario de una aplicación web Java. Los desarrolladores web escriben JSPs como archivos de texto que combinan código HTML o XHTML, elementos XML y acciones y comandos JSP incrustados.

Utilizando JSP, puede recopilar información de los usuarios a través de formularios de página web, presentar registros de una base de datos u otra fuente y crear páginas web de forma dinámica.

Las etiquetas JSP pueden ser utilizadas para una variedad de propósitos, tales como recuperar información de una base de datos o registrar preferencias de usuario, acceder a componentes JavaBeans, pasar el control entre páginas, y compartir información entre solicitudes, páginas, etc.

La página JSP es realmente otra forma de escribir un servlet. Excepto en la fase de traducción, una página JSP se maneja exactamente como un servlet normal. Sin embargo, el archivo JSP debe estar contenido en la carpeta WebContent.

```

<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri = "http://java.sun.com/jsp/jstl/functions" prefix = "fn" %>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>JSP Page</title>
</head>
<body>
<h3>La empresa ha sido añadido al registro</h3>

<p>Nombre: ${empresa.nombreEmpresa}</p>
<p>cif: ${empresa.cifEmpresa} </p>
<p>sector: ${empresa.sectorEmpresa} </p>
<p>Telefono: ${empresa.telefonoEmpresa}</p>
</body>
</html>

```

Jstl:librería de etiquetas para jsp.

Nota: hay que añadir la librería de Jstl para recorrer el objeto que hemos metido en la response previamente en el Servlet

## Servlets:

Un Servlet es un **objeto** java que pertenece a una clase que extiende de javax.servlet.http.HttpServlet

Son pequeños programas escritos en Java que admiten peticiones a través del protocolo HTTP. Los servlets reciben peticiones (**http request**) desde un navegador web, las procesan (o llaman a quien las procesa) y devuelven una respuesta (**http response**) al navegador. Para realizar estas tareas podrán utilizar las **clases** incluidas en el lenguaje Java. Estos programas son los intermediarios entre el cliente (casi siempre navegador web) y los datos (BBDD)

```

@WebServlet(name = "AddEmpresa", urlPatterns = {"/AddEmpresa"})
public class AddEmpresa extends HttpServlet {

    /**
     * Processes requests for both HTTP GET and POST
     * methods.
     *
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    private final String PATH = "resultAddEmpresaOk.jsp";
    @EJB
    EmpresaFacade misEmpresas;
    Empresa e;

    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");

        String nombreEmpresa = request.getParameter("nombreForm").trim();
        String cifEmpresa = request.getParameter("cifEmpresaForm").trim();
        String sectorEmpresa = request.getParameter("sectorEmpresaForm").trim();
        Integer telefonoEmpresa = Integer.parseInt(request.getParameter("telefonoEmpresaForm"));
        e = new Empresa(nombreEmpresa, cifEmpresa, sectorEmpresa, telefonoEmpresa);

        misEmpresas.create(e);
        request.setAttribute("empresa", e);
        RequestDispatcher dispatcher = request.getRequestDispatcher(PATH);
        dispatcher.forward(request, response); // utilizamos el dispatcher para añadir el objeto empresa a la request y poderlo leer en el jsp correspondiente
        // en este caso resultAddEmpresaOk.jsp
    }
}

```

## Dispatcher

Un **RequestDispatcher** es una clase extremadamente importante que permite "incluir" contenido en una request/response o "reenviar" una request/response a un recurso. Como ejemplo típico, un servlet puede utilizar un RequestDispatcher para incluir o reenviar una request/response a un JSP. En la programación de Modelo Vista Controlador en Java, un servlet normalmente sirve como el controlador. El controlador básicamente contiene o hace referencia al código para realizar determinadas acciones, y decide a que vista enviar al usuario. En JEE, la vista suele ser un JSP o un JSF. Se considera una mala práctica incluir código HTML en el propio servlet.

Se pueden establecer varios dispatchers en el mismo servlet, pero lógicamente solo se ejecutará uno. Estos dispatchers suelen incluirse bajo una cláusula de condicional y llevar consigo además valores como parámetros; variables primitivas, ArrayLists, objetos, etc. siempre que no superemos el límite de memoria de la Java Virtual Machine.

En este ejemplo se recoge un parámetro de la request, se añade a un carrito y se recoge el carrito actualizado en un ArrayList. A continuación se mete en la request tanto el artículo como el listado y se ejecuta el RequestDispatcher a un .jsp, donde se leerán esos parámetros para ser mostrados por pantalla.

```

protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    String articulo = (request.getParameter("articulo").toString());

    ArrayList<Articulo> listado = carrito.anadirArticulo(articulo);

    request.setAttribute("articulo", articulo);
    request.setAttribute("listado", listado);
}

```

```

RequestDispatcher dispatcher =
request.getRequestDispatcher("comprar.jsp");
dispatcher.forward(request, response);
}

```

Un **sendRedirect** envía una respuesta de redireccionamiento temporal al cliente utilizando la URL de la ubicación de redireccionamiento especificada y borra el buffer al crear una nueva request. Al llamar a este método, el código de estado se fija en 302 bajo GET, y 307 bajo POST. Es importante saber cuando utilizar dispatchers o redirects para evitar hackeos.

Este método puede aceptar URLs relativas; el contenedor servlet debe convertir la URL relativa a una URL absoluta antes de enviar la respuesta al cliente. Si la ubicación es relativa sin un ' / ' principal, el contenedor lo interpreta como relativo a la URI de solicitud actual. Si la ubicación es relativa con un ' / ' principal, el contenedor lo interpreta como relativo a la raíz del contenedor de servlets.

## Enterprise Java Beans(EJB).

Los EJBs son objetos Java ejecutados dentro de un entorno servidor.

- Los EJBs pueden distribuirse entre varias máquinas y se puede acceder a ellos de forma remota como si estuvieran en una máquina local.
- Los EJBs son manejados de forma centralizada por un contenedor. La relación entre un contenedor EJB y un EJB es conocida como "inversión de control" o el "Principio de Hollywood" (no nos llame, ya le llamaremos).

Un contenedor EJB es un entorno de ejecución para controlar Beans Enterprise. El contenedor almacena y maneja un Bean Enterprise de la misma manera que un motor servlet hospeda un servlet Java, y les proporciona servicios de bajo nivel. Son tan utilizados que tienen su propio import.

- Maneja las transacciones.
- Proporciona seguridad
- Proporciona persistencia
- Acceso remoto
- Control del ciclo de vida
- Repositorio de conexiones a la base de datos
- Repositorio de ejemplares del bean

Como el contenedor EJB maneja el paquete de servicios a nivel de infraestructura para un Bean Enterprise, un programador se puede concentrar en programar la lógica de negocio de ese Bean.



Una aplicación cliente que desee interactuar con un Bean Enterprise no accede directamente a él. En lugar de eso, el Bean está aislado de la aplicación cliente mediante el contenedor.

```
/* @author Iager
 */
@Stateless
@LocalBean
public class EmpresaFacade extends AbstractFacade<Empresa> implements EmpresaFacadeLocal {

    @PersistenceContext(unitName = "Crowdsolving.final1PU") // importante referenciar bien unidad persistencia
    private EntityManager em;

    @Override
    protected EntityManager getEntityManager() {
        return em;
    }

    public EmpresaFacade() {
        super(Empresa.class);
    }

    public Reto asignaReto(Long idEmpresa, Reto r) {

        Empresa emp = find(idEmpresa); // busqueda, proceso similar a HashMap
        List<Reto> retoemp = emp.getRetosEmpresa();
        retoemp.add(r); // se busca la empresa y luego se relaciona con el reto los retos se almacenan en ArrayList clase retos

        return r;
    }

    public void removeRetoToEmpresa(Long idReto) {

        for (Empresa empresa : findAll()) {
            List<Reto> susRetos = empresa.getRetosEmpresa();
        }
    }
}
```

## Session beans

Los session beans son objetos no persistentes que implementan la lógica del negocio que se ejecuta en el servidor. Es posible pensar que un session bean es una extensión lógica del programa cliente que se ejecuta en el servidor y contiene información específica al cliente. El tiempo de vida es limitado por la duración de la sesión del cliente, por lo cual no son persistentes en el tiempo.

Cada session bean mantiene una interacción con un cliente que se desarrolla a través de la ejecución de los distintos métodos que provee. Existen tres tipos de session beans que varían en la forma de modelar esta interacción: stateless, stateful y singleton.

Los stateless y stateful sirven a un cliente a la vez mientras que, por el contrario, los singleton pueden invocarse en concurrencia.

Debido a su tiempo de vida reducido y a la no persistencia de sus datos, un session bean no sobrevive a fallas en el container o en el servidor. En este caso el bean es eliminado de la memoria, siendo necesario que el cliente vuelva a iniciar una conexión para poder continuar con su uso.

Los session beans se incorporan al cliente mediante anotaciones o JNDI. Pueden a su vez implementar interfaces locales y remotas. Aquí no se tratará con interfaces y se seguirá la metodología moderna de anotaciones.

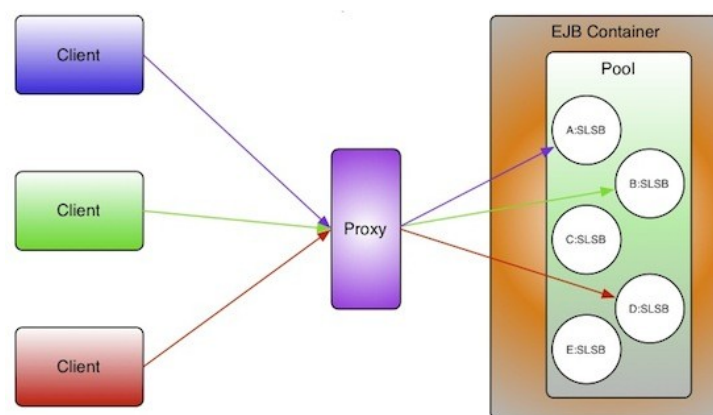
Un session bean debe estar siempre dentro de algún paquete. Si no lo elegimos a priori o no tenemos, Eclipse nos obligará a crear un paquete. Para simplificar los ejemplos, todos los archivos relacionados irán en el mismo paquete (menos los archivos de vista que van a WebContent como siempre).

## Stateless

Están pensados para modelar los procesos de negocios que tienden naturalmente a una única interacción, por tanto no requieren de mantener un estado entre múltiples invocaciones. Después de la ejecución de cada método, el container puede decidir mantenerlo, destruirlo, limpiar toda la información resultante de ejecuciones previas, o reutilizarlo en otros clientes. La acción a tomar depende de la implementación del container.

Los métodos independientes y que están determinados sólo por los parámetros entregados por el cliente son candidatos a ser representados en este tipo de bean. Por ejemplo un método que realice un cálculo matemático con los valores entregados y retorne el resultado, o un método que verifique la validez de un número de tarjeta de crédito son posibles métodos implementables por stateless session beans.

Debido a que no mantienen estado de interacciones, todas las instancias de la misma clase son equivalentes e indistinguibles para un cliente, sin importar quien utilizó un bean en el pasado. Por tanto pueden ser intercambiados entre un cliente y otro en cada invocación de un método, creando un pool de stateless session beans.

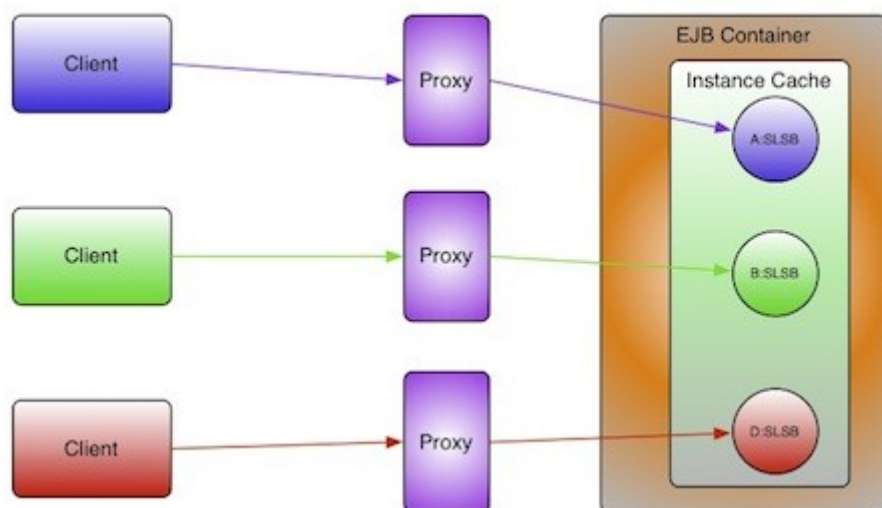


Sin embargo, es necesario señalar que sólo un cliente puede invocar una instancia a la vez, esto es, el container no debe permitir la ejecución de múltiples threads sobre una misma instancia para evitar posibles problemas derivados del procesamiento paralelo.

El siguiente **ejemplo** crea un .jsp que envía un número al servlet. El servlet llama al bean, el bean le devuelve la suma del número almacenado más el introducido y se hace un dispatcher al .jsp donde se ve el resultado. Al usar JSTL hay que meter la librería. Si por cualquier extraña razón marca error en el doGet y doPost, añade también la librería servlet-api, que está ya dentro de la carpeta del servidor TomEE.

## Stateful

Un stateful session bean preserva el estado conversacional con el cliente (servlet). Como su nombre indica mantiene asociados al cliente con sus variables de instancia. El contenedor EJB crea un bean separado para cada uno de los clientes. **Tan pronto como el scope de la solicitud haya terminado el stateful se destruirá.**



## Singleton

Proporcionan un constructor de programación formal que garantiza que una session bean se instanciará una vez por aplicación en una Máquina Java Virtual particular.(JVM), y que existirá para el ciclo de vida de la aplicación. Con los singletons, se puede compartir fácilmente el estado entre las instancias múltiples de un Enterprise bean o entre varios componentes de la aplicación.

Los singleton ofrecen una funcionalidad similar a los stateless, pero difieren de ellos en el sentido de que sólo hay un bean de sesión de por aplicación, en vez de un grupo de beans de sesión stateless donde cualquiera puede responder a una solicitud del cliente. Los beans singleton mantienen su estado entre las invocaciones de los clientes, el problema que teníamos en el stateful desaparece ahora con singleton.

## Patrones de programación

### Interface

Una interfaz no es más que una clase que tiene todos sus métodos sin implementar. Sólo con la definición del método. Aquellas clases que implementen a ésta, tendrán que definir la implementación de dichos métodos.

Difiere de una clase abstracta, en que en ésta última, puede haber métodos implementados y otros que no. Si una clase contiene uno o más métodos abstractos, está clase debe ser abstracta.

Las clases que hereden de la interfaz deben implementar **todos** sus métodos. El número de parámetros de los métodos, así como el tipo de dato de los mismos, deben respetarse o de lo contrario se está hablando de otro método totalmente diferente (sobrecarga de métodos).

En Java, una sub-clase puede heredar de sólo una super-clase pero puede implementar varias interfaces (herencia múltiple).

El concepto de una interfaz es fundamental en la mayoría de los lenguajes de programación orientados a objetos. En algunos, los objetos son conocidos sólo a través de sus interfaces de modo que no hay manera de acceder a un objeto sin tener que ir a través de ella, con lo cual debe establecerse como public.

Los usos más comunes para las interfaces son:

Deseamos describir el protocolo de comunicación de una clase fuera de su paquete.

Se debe cambiar la implementación de una funcionalidad en tiempo de ejecución.

Durante el diseño desconocemos cual será la implementación que se usará en tiempo de compilación.

```
public interface IPersonas {  
    public Persona anadirPersona(Persona persona);  
}
```

```

        public ArrayList<Persona> listarPersonas();
        public boolean eliminarPersona(String dni);
        public boolean pagar(String dni);
        public int buscarDni(String dni);
    }

```

Una clase que desee trabajar con los métodos de la interfaz debe implementar dicha interfaz:

```

public class FPersonas implements IPersonas{

```

## Facade

Se aplicará el patrón fachada cuando se necesite proporcionar una interfaz simple para un subsistema complejo, o cuando se quiera estructurar varios subsistemas en capas, ya que las fachadas serían el punto de entrada a cada nivel. Otro escenario proclive para su aplicación surge de la necesidad de desacoplar un sistema de sus clientes y de otros subsistemas, haciéndolo más independiente, portable y reutilizable (esto es, reduciendo dependencias entre los subsistemas y los clientes).

Los clientes (por ejemplo un servlet) envían peticiones a la clase Fachada, la cual las ejecuta y devuelve el resultado al cliente que lo invoca. Esta definición no difiere mucho del funcionamiento de un simple método. Podemos, de alguna manera, considerar la fachada como una clase que contiene la implementación de los métodos de un objeto. La definición de los métodos de la fachada puede estar a su vez en una interfaz.

La principal ventaja del patrón fachada consiste en que para modificar las clases de los subsistemas, sólo hay que realizar cambios en la interfaz/fachada, y los clientes pueden permanecer ajenos a ello. Además, y como se mencionó anteriormente, los clientes no necesitan conocer las clases que hay tras dicha interfaz.

Como inconveniente, si se considera el caso de que varios clientes necesiten acceder a subconjuntos diferentes de la funcionalidad que provee el sistema, podrían acabar usando sólo una pequeña parte de la fachada, por lo que sería conveniente utilizar varias fachadas más específicas en lugar de una única global.

```

public class FPersonas implements IPersonas{
    public ArrayList<Persona> listado;
    public FPersonas() {

```

```

        listado = Singleton.getInstance() }

public ArrayList<Persona> listarPersonas() {
    return listado;
}

public Persona anadirPersona(Persona persona) {
    String dni = persona.getDni();

    if (buscarDni(dni)<0) { // si el dni no existe ya.
        if (listado.add(persona)) {
            return persona;
        } else {
            return null; }
    }
}

```

```

public EmpresaFacade() {
    super(Empresa.class);
}

public Reto asignaReto(Long idEmpresa, Reto r) {

    Empresa emp = find(idEmpresa); // busqueda, proceso similar a HashMap
    List<Reto> retoemp = emp.getRetosEmpresa();
    retoemp.add(r); // se busca la empresa y luego se relaciona con el reto los retos se almac
    return r;
}

public void removeRetoToEmpresa(Long idReto) {

    for (Empresa empresa : findAll()) {
        List<Reto> susRetos = empresa.getRetosEmpresa();
        for (Reto reto : susRetos) {
            if (reto.getIdReto().equals(idReto)) {
                em.remove(reto);
            }
        }
    }
}

public void removeEmpresa(Long idEmpresa) {

    Empresa e = find(idEmpresa);
    em.remove(e); //primero busca la empresa y luego si la encuentra la borra
}

public Empresa modEmpresa(Long idEmpresa, String nombreEmpresa, String cifEmpresa,

```

## Singleton

El patrón Singleton forma parte de los patrones creacionales. Se trata de uno de los patrones más usados y conocidos por los desarrolladores, y también es uno de los patrones más controvertidos. El patrón Singleton se encarga de controlar que únicamente se crea una instancia de una clase en toda la aplicación mediante el uso de un único punto de acceso.

Sólo se proporciona un punto de acceso para crear una instancia de una clase Singleton. El constructor es privado y se proporciona un método **getInstance()** que se encarga de proporcionar el acceso a la clase. Cuando el método `getInstance()` es llamado por dos threads al mismo tiempo puede provocar problemas de concurrencia y permitir la creación de dos instancias de la clase, aunque esto es solucionable.

El problema que se resuelve con el Singleton en los ejercicios posteriores, es que la implementación del objeto en la fachada provoca que siempre que se acceda a esa fachada, el contenido se vacía debido a la reimplementación, devolviendo excepciones después del primer uso. En cambio, si en la fachada se obtiene la instancia creada en el Singleton podemos simular una persistencia.

El patrón Singleton tiene muchas ventajas sobre una clase estática:

Una clase Singleton puede extender e implementar interfaces, mientras que una clase estática no puede.

Una clase Singleton puede tener una inicialización tardía mientras que una clase estática tiene una inicialización temprana.

Aunque la principal ventaja es que una clase Singleton permite el polimorfismo sin forzar al usuario a asumir que sólo hay una instancia.

Alguna de las razones por las que no se debería utilizar sería:

Ocultas las dependencias: Un componente que usa un Singleton oculta información sobre dependencias. Cuando se exponen las dependencias fuerza a pensar en ellas y a pensar cuando debemos utilizar un componente.

Difícil de testear: El acoplamiento oculto hace que las pruebas sean muy complicadas, ya que no hay manera de inyectar una instancia de prueba del Singleton. Además, el estado del Singleton afecta a la ejecución de un conjunto de pruebas de tal manera que no están adecuadamente aislados entre sí.

```
public class Singleton {
    private static ArrayList<Persona> listado;

    public Singleton() {}

    public static ArrayList<Persona> getInstance() {
        if (listado == null) {
            listado = new ArrayList<Persona>();
        }
        return listado;
    }
}
```

En la fachada **obtenemos la instancia**:

```
public ArrayList<Coche> listarCoches() {  
    return listado = Singleton.getInstance();}
```

## ArrayList y HashMap:

Para almacenar los objetos de manera simulada en la instancia del Singleton se pueden utilizar varios formatos de listas y/o arrays, entre ellos el ArrayList y el HashMap. Ambas opciones nos proporcionan añadir objetos a un array sin necesidad de redimensionar el array. La diferencia fundamental es que el Hashmap utiliza un atributo del objeto insertado como índice de búsqueda y eliminación del propio objeto, con lo que no tiene que recorrerlo (en teoría).

En los ejercicios posteriores se trabajará con ArrayList para mantener los conceptos a lo largo de todos ellos. Si bien no es la opción óptima, para el nivel y la forma de uso que se le dará la consideraremos válida. El principal argumento en contra es que para encontrar el elemento siempre se recorre todo el array. Como usar break no es una opción ya que rompe el flujo de programa, el escape del bucle se hará igualando la variable índice a la longitud del array, cumpliéndose así la condición de salida del bucle.

```
ArrayList<Coche> listado = new ArrayList<Coche>();
```

### Añadir objeto:

```
listado.add(coche); // objeto de la misma clase que el ArrayList
```

### Buscar un atributo:

```
for(int i = 0; i < listado.size(); i++) {  
    if (listado.get(i).getMatricula().equalsIgnoreCase(matricula)) {  
        //i se ha encontrado!! (podemos guardar la posición donde se encontró)  
        i = listado.size(); // salida del bucle.  
    }  
    // y devolver la posición para otros métodos.  
}
```

### Obtener objeto a partir de atributo:

```
int posicion = -1; // se utiliza -1 porque 0 es una posición válida en un array.  
for(int i = 0; i < listado.size(); i++) {  
    if (listado.get(i).getMatricula().equalsIgnoreCase(matricula)) {  
        posicion = i;  
        i = listado.size();  
    }  
}  
if (posicion >= 0) {  
    return listado.get(posicion); // devuelve el objeto. get solo admite (int)
```



```

    }
    else {
        return null;
    }

```

### Modificar un atributo:

```

int posicion = buscarMatricula(matricula); // método no mostrado creado aparte.
if (posición >= 0 ) {
    listado.get(posicion).setColor(color);
}

```

### Eliminar un objeto: se puede pasar el índice a borrar o el propio objeto.

```

int posicion = buscarMatricula(matricula); // método no mostrado creado aparte.
if (posicion >= 0) {
    listado.remove(posicion); // elimina el objeto en la posición recibida
} // el objeto también podría ir como parámetro del remove().

```

### Vaciar el ArrayList:

```

listado.clear();

```

### Comprobar si está instanciado pero vacío:

```

listado.isEmpty();

```

### Utilizar un HashMap:

Declaración de un HashMap con clave Integer y Valor String. Las claves pueden ser de cualquier tipo de objetos, aunque los más utilizados como clave son los objetos predefinidos de Java como String, Integer, Double... Los Map no permiten datos atómicos.

```

Map<Integer, String> nombreMap = new HashMap<Integer, String>();

```

```

nombreMap.size(); // Devuelve el numero de elementos del Map
nombreMap.isEmpty(); // Devuelve true si no hay elementos en el Map y false si sí.
nombreMap.put(K clave, V valor); // Añade un elemento al Map
nombreMap.get(K clave); // Devuelve el valor de la clave que se le pasa o 'null'
nombreMap.clear(); // Borra todos los componentes del Map
nombreMap.remove(K clave); // Borra el par clave/valor de la clave que se le pasa.
nombreMap.containsKey(K clave); // Devuelve true si en el map hay una clave igual
nombreMap.containsValue(V valor); // Devuelve true si hay un valor que coincide con V
nombreMap.values(); // Devuelve una "Collection" con los valores del Map

```

Todo sobre los hashmap: <https://jarroba.com/map-en-java-con-ejemplos/>  
<https://www.youtube.com/watch?v=-JOSjIa2g0>

## Entity manager factory:

Una conexión a una base de datos está representada por una instancia **EntityManager**, que también proporciona funcionalidad para realizar operaciones en una base de datos. Muchas aplicaciones requieren múltiples conexiones a bases de datos durante su vida útil. Por ejemplo, en una aplicación web es común establecer una conexión de base de datos separada, usando una instancia de EntityManager separada para cada petición HTTP.

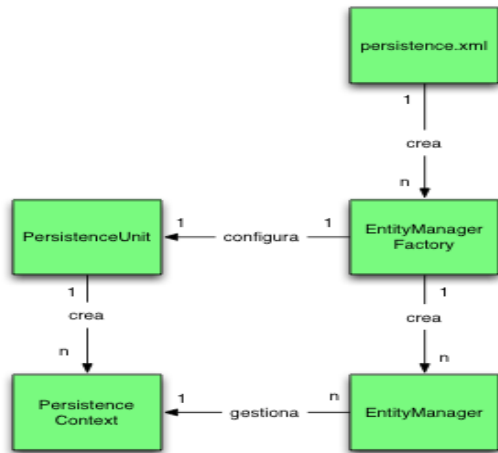
El papel principal de una instancia de **EntityManagerFactory** es apoyar la instanciación de instancias de EntityManager. Un EntityManagerFactory se construye para una base de datos específica, y mediante la gestión eficiente de los recursos (por ejemplo, un grupo de sockets), proporciona una forma eficiente de construir múltiples instancias de EntityManager para esa base de datos. Una vez construida, puede servir a toda la aplicación.

En un primer lugar un EntityManagerFactory es un generador único con el que nosotros gestionamos todas las entidades. Ahora bien si tenemos varias conexiones a base de datos deberemos definir un nuevo concepto que nos permite clarificar que tenemos dos EntityManagerFactories distintos. Este concepto es el que se conoce como **PersistenceUnit** (la unidad de persistencia explicada anteriormente). Cada PersistenceUnit tiene asociado un EntityManagerFactory diferente que gestiona un conjunto de entidades distinto.

Cuando una EntityManagerFactory se cierra, todos los EntityManagers de ese factory, y por extensión todas las entidades gestionadas por esos EntityManagers, pierden su validez.

Es mucho mejor mantener un factory abierto durante un largo período de tiempo que crear y cerrar repetidamente nuevos factory. Por lo tanto, la mayoría de las aplicaciones nunca cerrarán el factory, o solo la cerrarán cuando la aplicación está cerrando.

Solo se permite crear una EntityManagerFactory para cada configuración de unidad de persistencia desplegada. Se puede crear cualquier número de instancias de EntityManager desde un factory determinado.



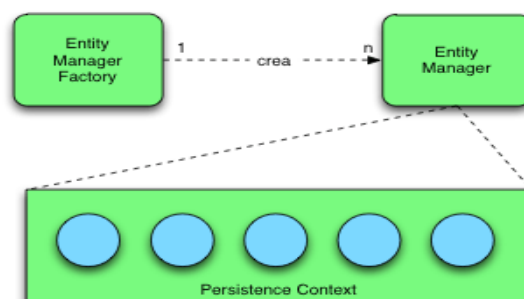
## Entity manager:

Es un gestor de recursos que mantiene la colección activa de objetos de entidad que está utilizando la aplicación. El EntityManager maneja la interacción y metadatos de bases de datos para las correlaciones relacionales de objetos. Una instancia de EntityManager representa un contexto de persistencia.

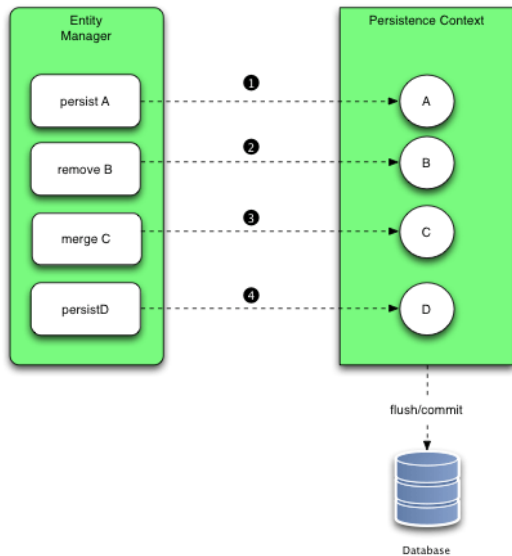
En el caso de ejecutar nuestro programa como una aplicación de escritorio será necesario instanciar el EntityManager a través de la clase EntityManagerFactory. El programador es el responsable de abrir y cerrar las transacciones del propio manager (**em**).

```
em.getTransaction().begin();
Empleado empleado = new Empleado();
empleado.setNombreEmpleado("Nombre");
empleado.setApellidosEmpleado("Apellido1 Apellido2");
em.persist(empleado);
em.getTransaction().commit();
em.close();
```

Cuando nuestra aplicación vive dentro de una Application Server o en un EJB Container, los EntityManager se referencian por medio de inyección de dependencias (CDI), utilizando la anotación **@PersistenceContext**, sin factory (se verá en la parte de persistencia siguiente). De este modo, el Application Server es el encargado de la creación de los EntityManager para su inyección. Un detalle interesante de este método es que el Application Server se puede encargar de forma automática de las transacciones, abriendo una transacción al inicio de la ejecución de un método de negocio y cerrando la transacción automáticamente al finalizar la invocación del método.



## PRINCIPALES MÉTODOS DEL ENTITY MANAGER



**Flush().** Este método es un poco curioso y es el encargado de sincronizar el `PersistenceContext` contra la base de datos. Normalmente todo el mundo piensa que cuando nosotros invocamos el método `persist()` o `remove()` se ejecutan automáticamente las consultas contra la base de datos. Esto no es así ya que el `EntityManager` irá almacenando las modificaciones que nosotros realizamos sobre las entidades para después persistirlas todas de golpe ya sea invocando `flush` o realizando un `commit` a una transacción. En algunos casos de todos modos, puedes querer que las instrucciones

SQL se ejecuten inmediatamente; generalmente cuando necesitas el resultado de alguna función secundaria, como una clave autogenerada o un trigger de base de datos.

¿Qué pasa si se lanza una excepción después de eso? ¿El entity manager lo echará todo a perder? ¿Incluso los datos escritos en la primera descarga? Puede hacer `rollback` la base de datos, pero no hará ningún cambio en los objetos, por ejemplo, 'versión' autoincrementado, ID autogenerado, etc. Además, el entity manager se cerrará después de un `rollback`. Si intentas "mergear" el objeto en otra sesión, la 'version' auto-incrementada en particular puede causar una `OptimisticLockException`.

Es complicado encontrarse ese tipo de situaciones en este nivel. Este apartado se incluye como información adicional debido a la cantidad de ejemplos que se pueden encontrar con un uso indiscriminado. Si persistimos a través del modo `CMP/JTA` no deberíamos tocar sus colas de trabajo a no ser que sepamos exactamente lo que estamos haciendo.

- **persist():** Hace una instancia de una entity gestionable y persistente.
- **find():** Busca por clave primaria. Busca una entidad de la clase y clave primaria especificadas. Si la instancia de entidad está contenida en el contexto de persistencia, se devuelve desde allí.
- **merge():** Fusiona el estado de la entidad dada en el contexto actual de persistencia (BD)

- **remove():** Elimina la entidad instanciada.
- **close():** Cierra el entity manager gestionado por aplicación. Después de que se haya invocado el método de cierre, todos los métodos en la instancia del EntityManager y cualquier objeto Query y TypedQuery obtenido de él arrojará la ExcepciónIllegalStateException, excepto getProperties, getTransaction e isOpen (que devolverá falso). Si se llama a este método cuando el gestor de la entidad está asociado a una transacción activa, el contexto de persistencia permanece gestionado hasta que la transacción se completa.
- **createNamedQuery():** Crea una instancia de Query para ejecutar una consulta determinada (en el lenguaje de consulta de persistencia de Java o en SQL nativo).
- **createNativeQuery():** Crea una instancia de Query para ejecutar una instrucción SQL nativa, por ejemplo, para actualizar o eliminar.

La instancia de Query tiene a su vez los siguientes métodos fundamentales: `getFirstResult()`, `getMaxResults()`, `getParameter()`, `getParameterValue()`, `getParameters()`, `getResultList()`, `getSingleResult()`, `setFirstResult()`, `setMaxResults()`, `setParameter()`...

- **getCriteriaBuilder():** Devuelve una instancia de CriteriaBuilder para la creación de objetos CriteriaQuery, que se utiliza para construir consultas de criterios, selecciones compuestas, expresiones, predicados, órdenes.

## Referencias:

<https://vladmihalcea.com/the-best-way-to-map-a-onetomany-association-with-jpa-and-hibernate/>

<http://www.thejavageek.com/2014/09/26/jpa-mapsid-example/>

<https://www.objectdb.com/api/java/jpa/MapsId>

<https://www.udemy.com/course/jsp-servlet-free-course/learn/lecture/5377396#overview>

<https://www.udemy.com/course/java-ee-jsf-ejb-jpa-web-services-seguridad-desde-cero-a-experto/learn/lecture/7621394#overview>

<https://www.udemy.com/course/java-tutorial/learn/lecture/172757#overview>

(este último, es el mejor curso para aprender a programar que conozco y es gratuito)

## **Mejoras de la aplicación:**

### **-Desplegar la aplicación en un servidor real:**

Para empezar necesito pagar un servidor (lo he incluido en los gastos).

Por ello se requiere para esta parte del proyecto la contratación de un programador senior con experiencia en despliegue de aplicaciones en entornos profesionales, para supervisar el proceso, porque tiene un gran riesgo por razones de seguridad.

Por esta razón lo he incluido en el presupuesto.

Estimamos un precio hora de programador senior de 35 eur/h.

Esto tiene la ventaja añadida de aprender como se hace de forma óptima para la siguiente ocasión.

### **-Mejorar el aspecto visual la aplicación:**

Esta parte también se ha decidido externalizarla a un desarrollador , se estima que 10 horas deberían ser suficientes para realizar la tarea.

En la industria del software la persona que hace el backend y el frontend son diferentes.

## **Conclusión:**

La industria del software tiene una gran demanda para desarrolladores en java EE.

Por ejemplo, me consta que el grupo Inditex y todas sus auxiliares utilizan esta tecnología.

Esta es la principal razón por la que he elegido esta tecnología para mi futuro profesional. He puesto todo por mi parte para alcanzar el mejor nivel posible para afrontar las entrevistas técnicas.

Por este motivo he elegido hacer un proyecto muy ambicioso, como es una plataforma web.

Las plataformas web en la industria son desarrolladas al menos por equipos de cinco personas.

Estoy orgulloso del resultado y quiero hacer incidencia en que he utilizado un amplio espectro de las herramientas que ofrece Java EE.



## WORK PLACE TUTOR OPINION LETTER

To whom it may concern:

As work place tutor of Andres Lage Freire I would like to express my opinion of his professional skills and knowledge. Andres have worked with me from October till December in 2019. My personal evaluation is that Andres is a hard working person with common sense for success in his professional future.

Under my supervision he was working on optimization algorithms coded in Java. Also he has developed a complete function database combined with Java EE web application. The complexity of the solution provided by Andres reaching the industrial standards for software products (especially in the back-end part of it).

I can measure the quality of the source code written by Andres as high level. The code is robust, clear, well documented and according wide established world wide standards in the field of software production. The front-end of the software solution can be improved form visual design point of view, but it does not make the achievement less successful.

During the period of supervision Andres was proactive in the communication with me. He took all my remarks seriously and I am glad that I saw how he progressed.

Sincerely,  
Dip.Eng. Todor Balabanov Ph.D.  
CEO of Velbazhd Software LLC  
Mladost 1 18 6 16  
1750 Sofia, Bulgaria  
+359 89 8237103



