GUÍA Nº 4 – MVC - HIBERNATE

FACULTAD	CURSO	AMBIENTE
INGENIERÍA	DISEÑO Y ARQUITECTURA DE SOFTWARE	LABORATORIO DE DISEÑO Y ARQUITECTURA DE SOFTWARE 77C0206

ELABORADO POR	ANÍBAL SARDÓN PANIAGUA	**APROBADO POR**	ARTURO RIVERA
VERSIÓN	001	**FECHA DE APROBACIÓN**	01/03/2022

1. LOGRO GENERAL DE UNIDAD DE APRENDIZAJE

El estudiante formula un informe descriptivo sobre los requerimientos de un producto de software para las necesidades de los Stakeholders, aplicando un lenguaje de modelado como UML y un proceso de desarrollo en la herramienta IBM RSA; definiendo la Visión del Negocio, modelo de casos de Uso y prototipos visuales.

2. OBJETIVOS ESPECÍFICOS DE LA PRÁCTICA

- Entender la Arquitectura MVC.
- Conocer lo componentes de la Arquitectura MVC y su implementación con JEE.
- Descubrir las tecnologías JEE para ORM.

3. MATERIALES Y EQUIPOS

- Computadoras Personales.
- Sistema Operativo Windows.
  Apache Netbeans
  Wildfly
- Pizarra
- Plumón
- Mota

4. PAUTAS DE SEGURIDAD

Las computadoras y laptops deben de estar prendidas mientras se usan. Pero al terminar el laboratorio estas deben dejarse apagadas.

- En el laboratorio debe estar prendido el aire acondicionado para evitar sobrecalentamientos y averías, especialmente en épocas de altas temperaturas.
- Los estudiantes no pueden llevar alimentos que puedan derramar sobre los computadores.
- Computadoras, router, switch, puntos de acceso (caídas).
- Eléctricos, por contacto directo o indirecto, electricidad estática y por fenómeno térmico. Puede producir: electrocuciones y quemaduras.
- Procedimiento ante Corte de Energía Eléctrica
- No tocar el equipo eléctrico en el que se encuentra trabajando, puede que retorne la energía.
- Comunicarse con el Asistente de Operaciones de turno quien se comunicará con el Técnico.

5. FUNDAMENTO

La asignatura de Diseño y Arquitectura de Software es de carácter teórico-práctico y tiene el propósito de potenciar en el estudiante sus habilidades para analizar y diseñar una arquitectura de software. Se desarrolla los siguientes contenidos: Introducción a la arquitectura de software, vistas y estilos de la arquitectura, requisitos de calidad de un software, diagramación UML orientada al diseño arquitectónico de software, patrones de arquitectura, arquitectura orientada a servicios (SOA), Arquitecturas en Cloud Computing , Arquitecturas para software en dispositivos móviles y documentación de una arquitectura de software.

6. INTRODUCCIÓN (MARCO TEÓRICO)

INTRODUCCIÓN A SERVLETS

¿Qué es un Servlet?

Es una clase escrita en lenguaje Java usada para extender las funcionalidades de un servidor mediante el paradigma request/response.

Aunque pueden responder a cualquier request, generalmente se usan para servidores web, para lo cual se definen clases HTTP específicas.

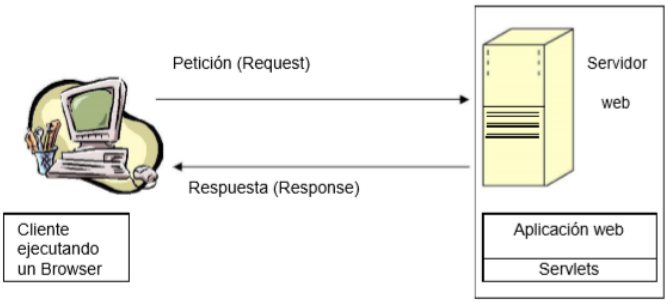
Arquitectura del Servlet

Un servlet es un componente JEE que extiende la capacidad de proceso de un servidor que emplea el paradigma request – response.

Por lo tanto, “Un servlet es una clase Java que recibe requerimientos de un cliente para cumplir con un servicio, luego de cumplir con el servicio, envía la respuesta hacia el cliente.”

La API Servlet se constituye de dos paquetes básicos:

- javax.servlet
- javax.servlet.http.



Ciclo de Vida de un Servlet

El ciclo de vida del servlet está compuesto de tres fases:

- El método init. Este método es llamado por el servidor de aplicaciones cuando el servlet se está cargando en memoria.
- El método service. Este método es llamado por cada petición de cliente. Para las peticiones HTTP, este método se ha especializado para enviar la petición al método doGet o doPost.
- El método destroy. Este método es llamado por el servidor de aplicaciones cuando el servlet es descargado de memoria.

Los Servlets son cargados en memoria en la primera petición de un cliente o cuando el servidor de aplicaciones arranca.

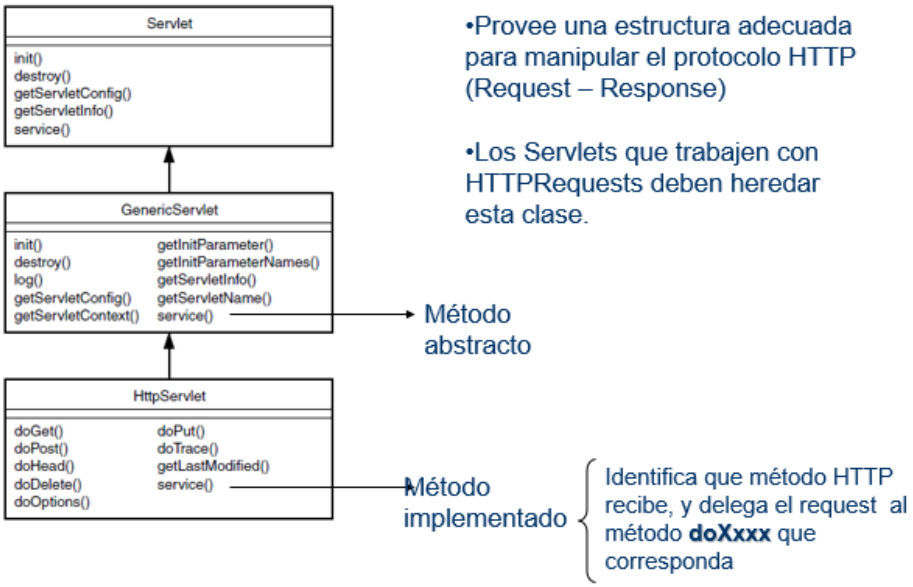
Cada petición de cliente es servida sobre un diferente *hilo (thread);* por lo tanto, muchos clientes pueden acceder al mismo código en paralelo.

La clase HTTPServlet

Los clientes web (browsers) interactúan con los Servlets usando el protocolo HTTP (Request-Response)

Por lo tanto, para crear Servlets que soporten este protocolo, se debe heredar de la clase javax.servlet.http.HttpServlet.

La clase HttpServlet provee una estructura de trabajo adecuada para manipular el protocolo HTTP junto con los métodos GET y POST



Despachador de la petición (Request Dispatcher)

El despachador de la petición (RequestDispatcher) permite remitir una petición (request) a otro servlet o a otro componente JEE.

Un objeto RequestDispatcher es obtenido de la siguiente manera:

```
RequestDispatcher rd = getServletContext ().getRequestDispatcher ("ruta");
```

Una vez obtenido el objeto RequestDispatcher, invocamos al otro recurso (servlet) enviándole el objeto request.

```
rd.forward (request, response)
```

Hay que registrar el servlet MyForwardServlet para lo cual debemos modificar el archivo web.xml.

INTRODUCCIÓN A PAGINAS JAVA SERVER PAGE (JSP)

JSP permite la creación de sitios web dinámicos donde se puede combinar contenido HTML y bloques de programación en java que permitan la programación de sentencias.

JSP scripting incluye los siguientes elementos de código Java que pueden aparecer dentro de una página JSP:

- Scriptlets
- Expresiones.

Scriptlets

Son porciones de código Java mezclados con el lenguaje de marcas de la página.

Un scriptlet, o fragmento de código, puede consistir de una o más líneas de código Java y se puede usar con el código HTML o JSP para configurar saltos condicionales o bucles por ejemplo.

Un scriptlet JSP está contenido dentro de los símbolos **<%...%>** y se escribe usando la sintaxis Java.

Expresiones

Se evalúan y convierten a valores de String y luego son mostradas en la ubicación respectiva dentro de la página.

Una expresión JSP no termina en **";".**

Está contenida dentro de los símbolos: **<%=...%>.**

Objeto request

El objeto request está disponible de forma automática por lo que no es necesario instanciarlo.

getParameter(String), permite recuperar una variable enviada por *url.*

setAttribute(String, Object), para cargar *(setear)* el objeto en el *request*.

getAttribute(String), para recuperar el objeto

PATRON MVC

Problema

Las aplicaciones deben estar creadas sobre una arquitectura sólida y estable que facilite su implementación y permita realizar los cambios y expansiones de la aplicación, además de facilitar la modificación y adición de nuevas funcionalidades.

La aplicación debe estar creada de manera modular, debe estar conformado por diferentes módulos independientes. Esto permite también que la aplicación sea más fácil de mantener, ya que los cambios en una parte causarán un menor impacto en las demás partes al estar aisladas unas de otras.

Solución

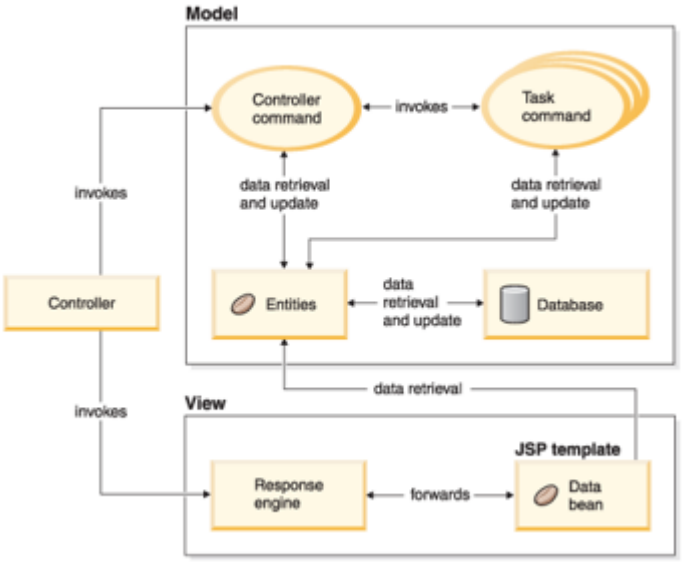
Como la mayor parte del costo de cada llamada está relacionado con la comunicación de ida y vuelta entre el cliente y servidor, una forma de reducir el número de llamadas es usando un objeto (el DTO) que agrega los datos que habrían sido transferidos por cada llamada, pero que son entregados en una sola llamada.

MVC separa los datos y la lógica de negocio de una aplicación de su representación y el módulo encargado de gestionar los eventos y las comunicaciones.

MVC propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador.

Por un lado define componentes para la representación de la información, y por otro lado para la interacción del usuario.

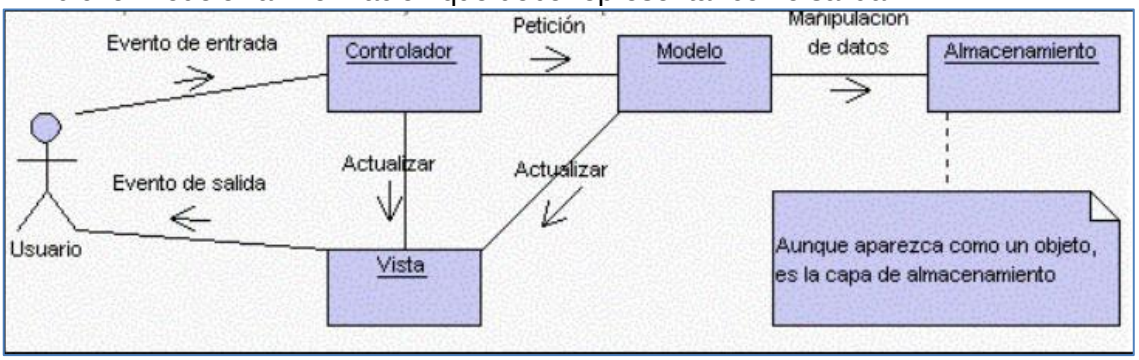
Se basa en las ideas de reutilización de código y la separación de conceptos, que buscan facilitar el desarrollo de aplicaciones y su posterior mantenimiento.

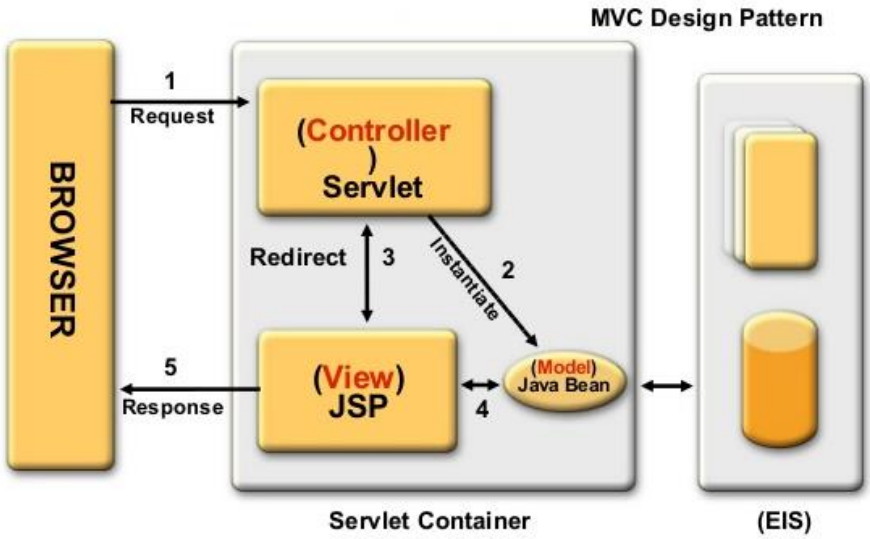


El Modelo: Es la representación de la información con la cual el sistema opera, por lo tanto gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación (lógica de negocio).

El Controlador: Responde a eventos (usualmente acciones del usuario) e invoca peticiones al 'modelo' cuando se hace alguna solicitud sobre la información.

La Vista: Presenta el 'modelo' (información y lógica de negocio) en un formato adecuado para interactuar (usualmente la interfaz de usuario), por tanto requiere de dicho 'modelo' la información que debe representar como salida.

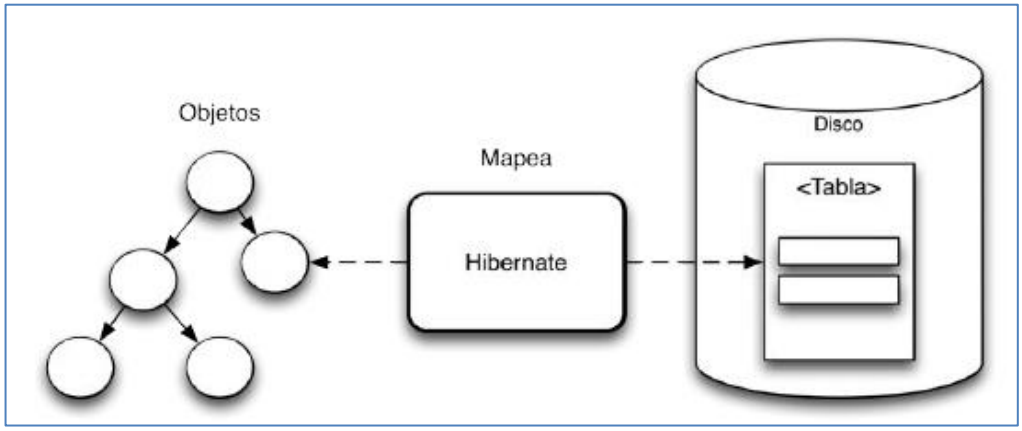


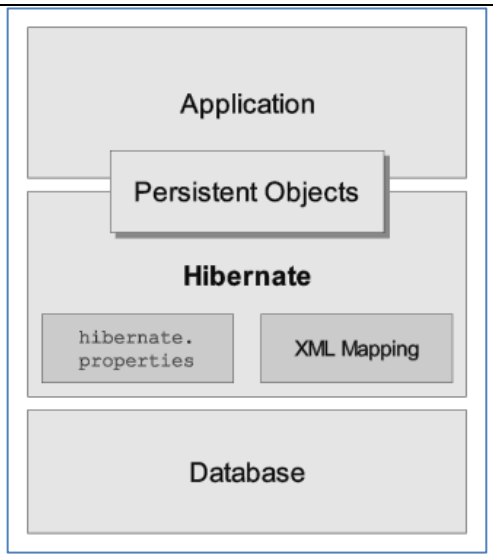


INTRODUCCION A HIBERNATE

Hibernate es una herramienta de **mapeo objeto-relacional (ORM)** para la plataforma Java que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos (XML) o anotaciones en los beans de las entidades que permiten establecer estas relaciones.

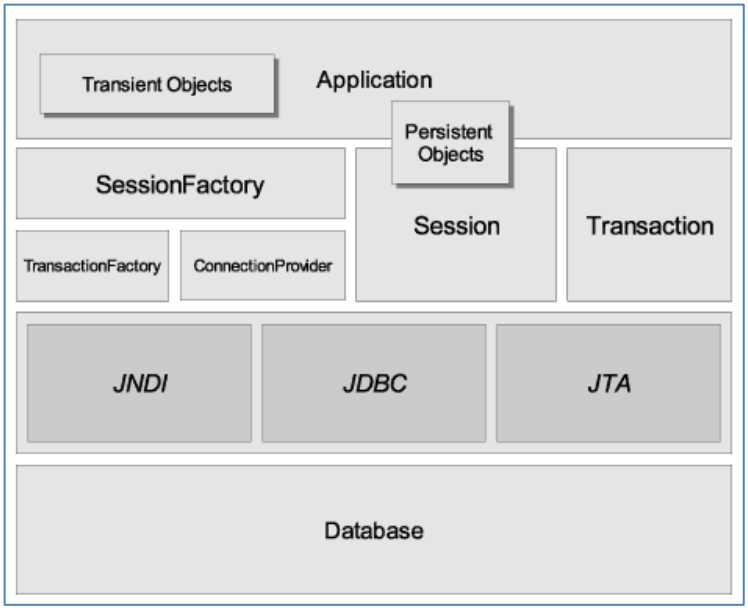
Hibernate es un framework **ORM (Object Relational Mapping)** cuya tarea es la de permitir a un desarrollador mapear objetos contra registros de una base de datos.





Se crea una capa entre la base de datos y la aplicación. Carga los detalles de la configuración, como la cadena de conexión de base de datos, clases de entidad, asignaciones, etc

Hibernate crea objetos persistentes que sincronizan los datos entre la aplicación y la base de datos.



Hibernate crear una instancia de clase de entidad *(clase Java mapeado con la capa de base de datos).* Este objeto se llama **objeto transitorio**, ya que aún no están asociados con la sesión o no conservan en una base de datos.
Para guardar el objeto de base de datos, se crea la instancia de la interfaz **SessionFactory**. **SessionFactory** es una instancia singleton que implementa el patrón de diseño de Factory. Cargas SessionFactory **hibernate.cfg.xml** archivo *(archivo de configuración de Hibernate)* y con la ayuda de **TransactionFactory y ConnectionProvider** implementa todos los ajustes de configuración en una base de datos.

Cada conexión de base de datos en Hibernate se crea mediante la creación de una instancia de la interfaz Session.
Sesión representa una única conexión con la base de datos.
Objetos de sesión se crean a partir de objetos SessionFactory.

Objeto de Configuración Hibernate

El objeto de configuración es el primer objeto de Hibernate que se crea en cualquier aplicación que utilice Hibernate y generalmente se crea una sola vez durante la inicialización de la aplicación. Representa un archivo de configuración o propiedades requeridas por Hibernate. El objeto de configuración proporciona dos componentes claves:

- **Conexión de base de datos:**
  Esto se maneja a través de uno o más archivos de configuración soportadas por Hibernate. Estos archivos son **hibernate.properties** y **hibernate.cfg.xml.**

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

    <session-factory>
        <!-- Database connection settings -->
        <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
        <property name="connection.url">jdbc:mysql://localhost/prueba</property>
        <property name="connection.username">el usuario</property>
        <property name="connection.password">la password</property>

        <!-- JDBC connection pool (use the built-in) -->
        <property name="connection.pool_size">1</property>

        <!-- SQL dialect -->
        <property name="dialect">org.hibernate.dialect.MySQLDialect</property>

        <!-- Enable Hibernate's automatic session context management -->
        <property name="current_session_context_class">thread</property>

        <!-- Disable the second-level cache  -->
        <property name="cache.provider_class">org.hibernate.cache.NoCacheProvider</property>

        <!-- Echo all executed SQL to stdout -->
        <property name="show_sql">true</property>

        <!-- Drop and re-create the database schema on startup -->
        <property name="hbm2ddl.auto">create</property>

        <mapping resource="com/chuidiang/ejemplos/hibernate/ejemplo1/Person.hbm.xml" />
    </session-factory>

</hibernate-configuration>
```

- **Configuración de Mapeo Clase:**
  Este componente crea la conexión entre las clases de Java y tablas de la base de datos.

```
"http://hibernate.sourceforge.net/hibernate-mapping-2.0.dtd">
<hibernate-mapping>
 <class name="entidades.Usuario" table="Usuario">
    <id name="id" type="int" column="UsuarioId">
      <generator class="native"/>
    </id>
    <property name="nombre" type="string"/>
    <property name="primerApellido" column="apellido1" type="string"/>
    <property name="segundoApellido" column="apellido2" type="string"/>
    <property name="direccion" type="string"/>
    <property name="telefono" type="int"/>
  </class>
</hibernate-mapping>
```

Objeto SessionFactory:

Objeto de configuración se utiliza para crear un objeto SessionFactory para la aplicación que utiliza el archivo de configuración suministrada y permite un objeto Session ser ejecutado. El SessionFactory es un objeto seguro para subprocesos y utilizado por todos los hilos de una aplicación.

El SessionFactory es objeto pesado por lo que normalmente se crea durante la aplicación puesta en marcha y mantenido para su uso posterior. Tendrían un objeto SessionFactory por la base de datos utilizando un archivo de configuración independiente. Si estamos utilizando múltiples bases de datos, entonces tendríamos que crear varios objetos SessionFactory.

```java
package com.chuidiang.ejemplos.hibernate.ejemplo1;

import java.io.File;

import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class HibernateUtil {
    private static final SessionFactory sessionFactory;
    static {
        try {
            // Create the SessionFactory from hibernate.cfg.xml
            sessionFactory = new Configuration().configure(new File("hibernate1.cfg.xml"))
                    .buildSessionFactory();
        } catch (Throwable ex) {
            // Make sure you log the exception, as it might be swallowed
            System.err.println("Initial SessionFactory creation failed." + ex);
            throw new ExceptionInInitializerError(ex);
        }
    }

    public static SessionFactory getSessionFactory() {
        return sessionFactory;
    }
}
```

Objeto Session:

Una Session se utiliza para obtener una conexión física con una base de datos. El objeto Session es ligero y está diseñado para ejecutarse cada vez que se necesita una interacción con la base de datos. Objetos persistentes se guardan y se recuperan a través de un objeto Session.

Los objetos de sesión no deben mantenerse abiertas durante mucho tiempo, ya que no suelen ser seguros para subprocesos y deben ser creados y destruidos, según sea necesario.

```java
Session session = factory.openSession();
Event e = (Event) session.load(Event.class, myEventId);
e.setName("New Event Name");
session.saveOrUpdate(e);
// later, with the same Session instance
Event e = (Event) session.load(Event.class, myEventId);
e.setDuration(180);
session.saveOrUpdate(e);
session.flush();
```

Objeto Transaction:

Una transacción representa una unidad de trabajo con la base de datos y la mayoría de los RDBMS soporta la funcionalidad de transacción. Las transacciones en Hibernate son manejados por un administrador de base de transacciones y las transacciones (de JDBC o JTA).

Este es un objeto opcional y las aplicaciones de Hibernate puede optar por no utilizar esta interfaz, en lugar gestionar las transacciones en su propio código de la aplicación.

```
Session session = factory.openSession();
Transaction tx = session.beginTransaction();
Event event = new Event();
// ... populate the Event instance
session.saveOrUpdate(event);
tx.commit();
```

Objeto Query:

Objetos de consulta utilizan **SQL** o **Hibernate Query Language (HQL)** cadena para recuperar datos de la base de datos y crear objetos. Una instancia de consulta se utiliza para enlazar los parámetros de consulta, limitar el número de resultados devueltos por la consulta, y finalmente, para ejecutar la consulta.

HQL (Hibernate Query Language)

Las consultas en Hibernate se encuentran estructuradas de forma similar al tradicional
SQL con las típicas claúsulas SELECT, FROM y WHERE.

```
Query q = session.createQuery("from Event");
List results = q.list();
```

```
Query q = session.createQuery("from Event");
q.setMaxResults(15);
List results = q.list();
```

```
Query q = session.createQuery("from Event where name = ? ");
q.setParameter(0, "Opening Plenary");
List results = q.list();
```

```
q.setParameter(0, "Opening Plenary", Hibernate.STRING);
```

```
Query q = session.createQuery("from Event where name = :name");
q.setParameter("name", "Opening Plenary");
List results = q.list();
```

Existe otro mecanismo muy interesante llamado named parameters (:nombreParametro)

```
Query q = session.createQuery("from Event where "+
    "startDate = :startDate or endDate < :startDate");
q.setParameter("startDate", eventStartDate);
List results = q.list();
```

CLAÚSULAS HQL

Clausula FROM

```
from Event e where e.name='Opening Plenary'
```

Clausula JOIN

```
from Event e join e.attendees a where a.id=314
```

Clausula SELECT

```
select e.name from Event e
```

```
select e.name, e.startDate from Event e
```

```
Session session = factory.openSession();
String query = " select e.name, e.startDate from Event e ";
Query query = session.createQuery("query");
List results = query.list();
for (Iterator I = results.iterator(); i.hasNext();) {
    Object[] row = (Object[]) i.next();
    String name = (String) row[0];
    Date startDate = (Date) row[1];
    // ...
```

Objeto Criteria:

Los objetos Criteria se utilizan para crear y ejecutar consultas con objetos y recuperar objetos.

7. PROCEDIMIENTO (DESARROLLO DE LA PRÁCTICA)

Ejemplo 01: Creación de Aplicación con una Arquitectura MVC con JEE e Hibernate.

1. Creación de la Base de Datos

Conectarse al servidor **MySQL** y ejecutar el siguiente **Script (Colegio):**

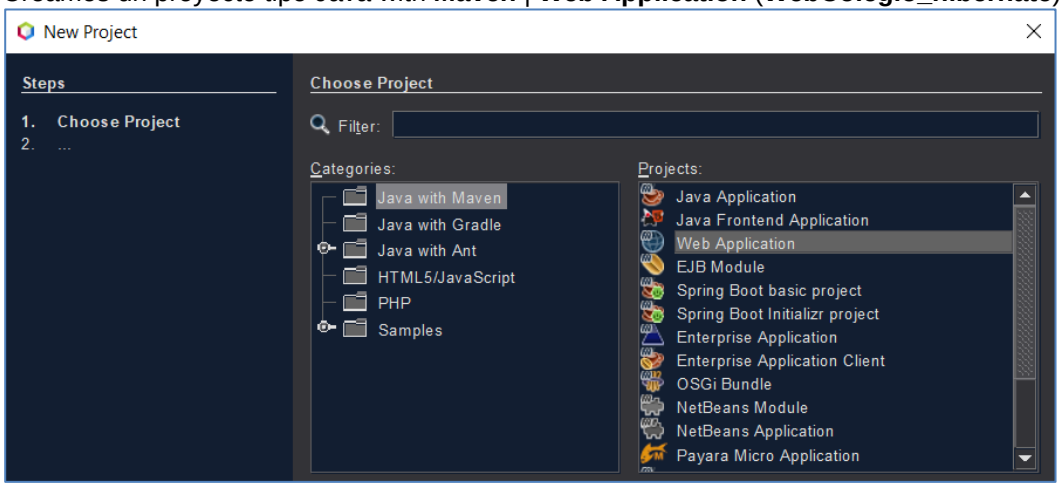
```
CREATE DATABASE  IF NOT EXISTS `colegio` /*!40100 DEFAULT CHARACTER SET utf8 */;
USE `colegio`;
-- MySQL dump 10.13  Distrib 5.6.24, for Win64 (x86_64)
--
-- Host: localhost    Database: colegio
-- ------------------------------------------------------
-- Server version    5.6.26-log

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8 */;
/*!40103 SET @OLD_TIME_ZONE=@@TIME_ZONE */;
/*!40103 SET TIME_ZONE='+00:00' */;
/*!40014 SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;
/*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0 */;
/*!40101 SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;
/*!40111 SET @OLD_SQL_NOTES=@@SQL_NOTES, SQL_NOTES=0 */;

--
-- Table structure for table `curso`
--

DROP TABLE IF EXISTS `curso`;
```

```
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `curso` (
`idCurso` int(11) NOT NULL AUTO_INCREMENT,
`nombre` varchar(45) DEFAULT NULL,
`docente` varchar(45) DEFAULT NULL,
`lugar` varchar(45) DEFAULT NULL,
PRIMARY KEY (`idCurso`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `curso`
--

LOCK TABLES `curso` WRITE;
/*!40000 ALTER TABLE `curso` DISABLE KEYS */;
INSERT INTO `curso` VALUES (1,'Matematicas','Carlos Hurtado','Sede Arequipa'),(2,'Lenguaje','Luis Felipe','Sede
Arequipa');
/*!40000 ALTER TABLE `curso` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `tbAlumno`
--

DROP TABLE IF EXISTS `tbAlumno`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `tbAlumno` (
`idtbAlumno` int(11) NOT NULL AUTO_INCREMENT,
`nombre` varchar(45) DEFAULT NULL,
`apellido` varchar(45) DEFAULT NULL,
`edad` int(11) DEFAULT NULL,
PRIMARY KEY (`idtbAlumno`)
) ENGINE=InnoDB AUTO_INCREMENT=9 DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `tbAlumno`
--

LOCK TABLES `tbAlumno` WRITE;
/*!40000 ALTER TABLE `tbAlumno` DISABLE KEYS */;
INSERT INTO `tbAlumno` VALUES
(1,'Luciana','Carpio',18),(2,'Carlos','Segovia',25),(7,'Luis','Garcia',21),(8,'Roxana','Zevallos',31);
/*!40000 ALTER TABLE `tbAlumno` ENABLE KEYS */;
UNLOCK TABLES;

--
-- Table structure for table `tdAdministrativo`
--

DROP TABLE IF EXISTS `tdAdministrativo`;
/*!40101 SET @saved_cs_client     = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `tdAdministrativo` (
`cod_emp` int(11) NOT NULL AUTO_INCREMENT,
`nom_emp` varchar(30) DEFAULT NULL,
`ape_emp` varchar(30) DEFAULT NULL,
`login_emp` varchar(20) DEFAULT NULL,
`clave_emp` varchar(15) DEFAULT NULL,
PRIMARY KEY (`cod_emp`)
) ENGINE=InnoDB AUTO_INCREMENT=9 DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `tdAdministrativo`
--

LOCK TABLES `tdAdministrativo` WRITE;
/*!40000 ALTER TABLE `tdAdministrativo` DISABLE KEYS */;
INSERT INTO `tdAdministrativo` VALUES
(1,'Carlos','Macedo','cmacedo','123456'),(2,'Luis','Carpio','lcarpio','654321'),(7,'Arturo','Delgado','adelgado','12345');
/*!40000 ALTER TABLE `tdAdministrativo` ENABLE KEYS */;
UNLOCK TABLES;
UNLOCK TABLES;
```

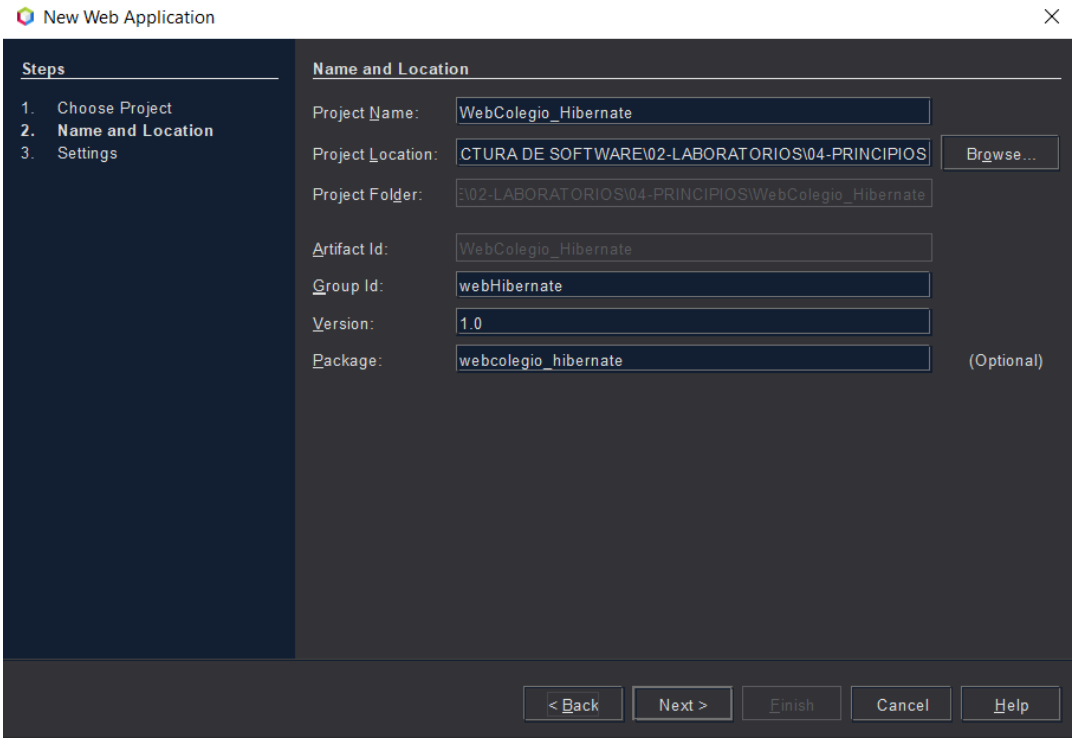
```
/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;

/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;
```

2. Creación del Proyecto

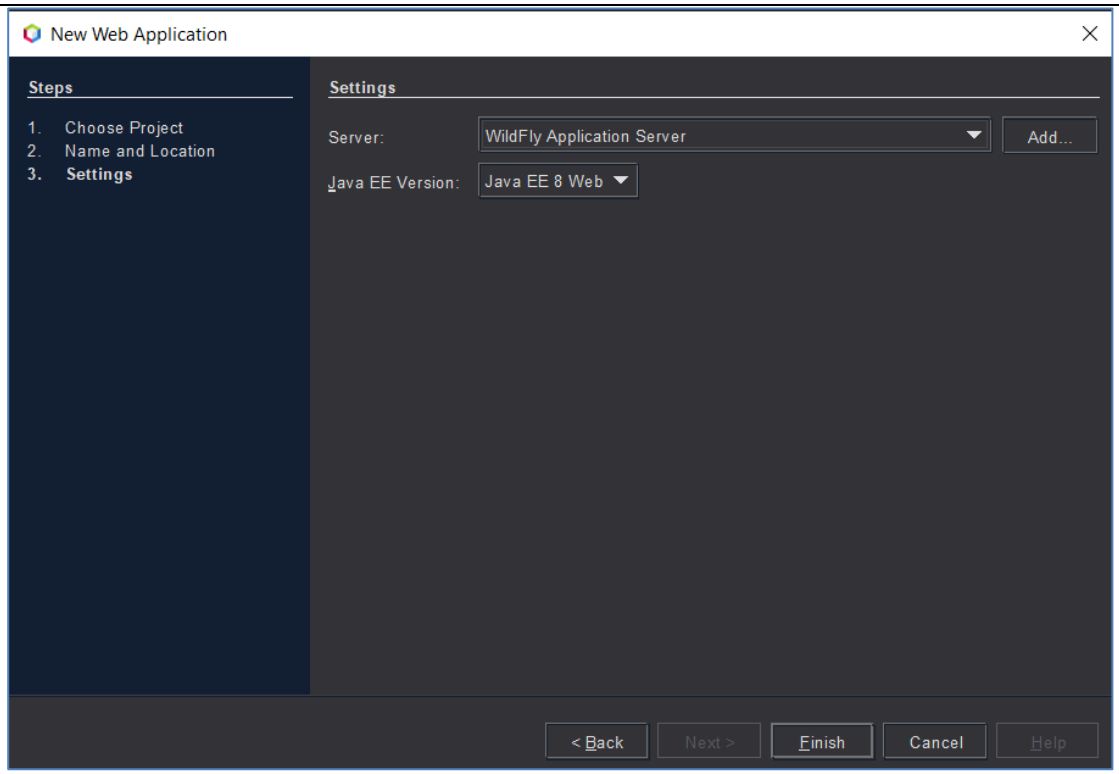
Creamos un proyecto tipo Java with **Maven** | **Web Application** (**WebColegio_hibernate**)



Ingresamos el nombre del proyecto y la ubicación



Seleccionamos el **Servidor Web**, la versión de **Java**



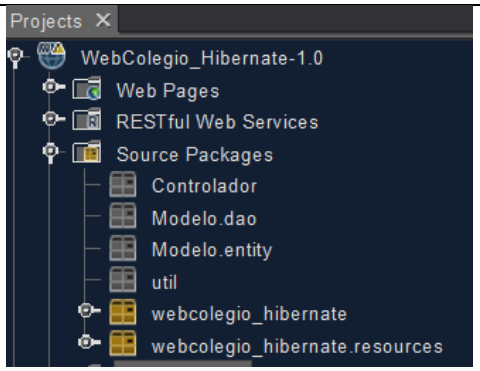
Ingresamos las **dependencias** para trabajar con **hibernate y MySQL**
Abrimos el archivo **POM** del **proyecto Maven** y agregamos las siguientes dependencias:

```
<dependencies>
    <dependency>
        <groupId>javax</groupId>
        <artifactId>javaee-api</artifactId>
        <version>${jakartaee}</version>
        <scope>provided</scope>
    </dependency>
    <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
     <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.0.21</version>
     </dependency>
     <!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->
    <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-core</artifactId>
            <version>5.4.23.Final</version>
    </dependency>
</dependencies>
```

3. Creación de los elementos del MVC

3.1 Vamos a crear las carpetas del proyecto.

En la carpeta **Source Package** agregaremos las siguientes paquetes:



3.2 En el paquete **util** crearemos la clase **HibernateUtil.**

Este archivo llama a las clases del framework Hibernate, contiene los elementos para configuración de hibernate y sirve para acceder al repositorio que será la base de datos.

```
package util;

import Modelo.entity.*;
import java.util.Properties;

import org.hibernate.SessionFactory;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
import org.hibernate.cfg.Configuration;
import org.hibernate.cfg.Environment;
import org.hibernate.service.ServiceRegistry;
/**
 * Versión 1.0
 * @author Ing. Aníbal Sardón Paniagua
 */
public class HibernateUtil
{
    private static SessionFactory sessionFactory;
    public static SessionFactory getSessionFactory()
    {
        if (sessionFactory == null)
        {
            try
            {
                Configuration configuration = new Configuration();

                // Hibernate settings equivalent to hibernate.cfg.xml's properties
                Properties settings = new Properties();
                settings.put(Environment.DRIVER, "com.mysql.cj.jdbc.Driver");
                settings.put(Environment.URL, "jdbc:mysql://localhost:3306/colegio?useSSL=false");
                settings.put(Environment.USER, "root");
                settings.put(Environment.PASS, "");
                settings.put(Environment.DIALECT, "org.hibernate.dialect.MySQL5Dialect");

                settings.put(Environment.SHOW_SQL, "true");

                settings.put(Environment.CURRENT_SESSION_CONTEXT_CLASS, "thread");

                settings.put(Environment.HBM2DDL_AUTO, "update");

                configuration.setProperties(settings);

                configuration.addAnnotatedClass(Curso.class);

                ServiceRegistry serviceRegistry = new StandardServiceRegistryBuilder()
                        .applySettings(configuration.getProperties()).build();

                sessionFactory = configuration.buildSessionFactory(serviceRegistry);
```

```
48                  }
                    catch (Exception e)
50                  {
                        e.printStackTrace();
52                  }
53              }
54              return sessionFactory;
55          }
56      }
```

3.3 Crearemos la **Entidad Curso**.

La Entidad representa la clase Persistente que se guardará en la Base de Datos y Tabla determinada.
Esta clase permite realizar el **mapeo** de la Entidad versus la Tabla, mediante las anotaciones

@Entity, el nombre la Clase Entidad en la aplicación.
@Table, el nombre de la Tabla en la Base de Datos.
@Column, el nombre de la Columna en la Tabla
@Id, el nombre de la Primary Key en la Tabla

```
1   package Modelo.entity;
2
3   import javax.persistence.Column;
4   import javax.persistence.Entity;
5   import javax.persistence.GeneratedValue;
6   import javax.persistence.GenerationType;
7   import javax.persistence.Id;
8   import javax.persistence.Table;
9   /**
10   * Versión 1.0
11   * @author Ing. Aníbal Sardón Paniagua
12   */
13  @Entity
14  @Table(name = "curso")
15  public class Curso
16  {
17          @Id
18          @GeneratedValue(strategy = GenerationType.AUTO)
19          @Column(name = "idCurso")
20          int idCurso;
21          @Column(name = "nombre")
22          String nombre;
23          @Column(name = "docente")
24          String docente;
25          @Column(name = "lugar")
26          String lugar;
27
28          public Curso(int idCurso, String nombre, String docente, String lugar) {
29              super();
30              this.idCurso = idCurso;
31              this.nombre = nombre;
32              this.docente = docente;
33              this.lugar = lugar;
34          }
```

```
35
36          public Curso()
37          {
38              super();
39          }
40          public int getIdCurso() {
41              return idCurso;
42          }
43          public void setIdCurso(int idCurso) {
44              this.idCurso = idCurso;
45          }
46          public String getNombre() {
47              return nombre;
48          }
49          public void setNombre(String nombre) {
50              this.nombre = nombre;
51          }
52          public String getDocente() {
53              return docente;
54          }
55          public void setDocente(String docente) {
56              this.docente = docente;
57          }
58          public String getLugar() {
59              return lugar;
60          }
61          public void setLugar(String lugar) {
62              this.lugar = lugar;
63          }
64      }
65
```

3.4 Crearemos las Clases DAO del **Modelo**.

En el paquete **modelo.dao** crearemos la **interface** y la **clase** que implementará la interfaz.

La interface define todos los métodos que contendrá la clase DAO.

```
1   package Modelo.dao;
2
3   import Modelo.entity.Curso;
4   import java.util.List;
5
6   /**
7    * Versión 1.0
8    * @author Ing. Aníbal Sardón Paniagua
9    */
10  public interface ICursoDAO
11  {
12      public void grabarCurso(Curso objCurso) ;
13      public void insertarCurso() ;
14      public void actualizarCurso(Curso objCurso) ;
15      public void eliminarCurso(int id) ;
16      public Curso obtenerCurso(int id) ;
17      public List<Curso> obtenerCursos() ;
18  }
19
```

La **clase DAO** llama a las **clases** de **Hibernate** para realizar las operaciones con el repositorio de datos.

Permite realizar:
- **Consultar objetos**, lanza una consulta al repositorio y retorna una lista de objetos/entidades.
- **Consultar un objeto**, lanza una consulta al repositorio y retorna un objeto/entidad.
- **Crear entidad (insertar),** inserta un registro en el repositorio.
- **Actualizar**, actualiza los datos de una entidad en el repositorio.
- **Eliminar**, elimina los datos de una entidad del repositorio.

En esta sección llama a las clases de hibernate.

```
1   package edu.utp.arquitectura.modelo.dao;
2
3   import edu.utp.arquitectura.modelo.entity.Curso;
4   import java.io.Serializable;
5   import java.util.List;
6
7   import javax.persistence.Query;
8
9   import org.hibernate.HibernateException;
10  import org.hibernate.Session;
11  import org.hibernate.Transaction;
12
13  import edu.utp.arquitectura.util.*;
```

Este método recibe una nueva entidad y la graba en el repositorio (inserta un registro).

```
14
15  public class CursoDAO implements ICursoDAO
16  {
17      @Override
        public void grabarCurso(Curso objCurso)
19      {
20          Transaction transaction = null;
21          try (Session session = HibernateUtil.getSessionFactory().openSession())
22          {
23              // start a transaction
24              transaction = session.beginTransaction();
25
26              // operation 1
27              Object object = session.save(objCurso);
28
29              // operation 2
30              session.get(Curso.class, (Serializable) object);
31
32              // commit transaction
33              transaction.commit();
34          }
35          catch (Exception e)
36          {
37              if (transaction != null)
38              {
39                  transaction.rollback();
40              }
                e.printStackTrace();
42          }
43      }
```

Este método recibe los datos de nueva entidad y graba los datos en el repositorio (inserta un registro).

```java
@Override
public void insertarCurso()
{
    Transaction transaction = null;
    try (Session session = HibernateUtil.getSessionFactory().openSession())
    {
        // start a transaction
        transaction = session.beginTransaction();

        String hql = "INSERT INTO Curso (idCurso, nombre, docente, lugar) "
                   + "SELECT idCurso, nombre, docente, lugar FROM Curso";


        Query query = session.createQuery(hql);


        int result = query.executeUpdate();
        System.out.println("Rows affected: " + result);

        // commit transaction
        transaction.commit();
    }
    catch (HibernateException e)
    {
        if (transaction != null)
        {
            transaction.rollback();
        }
        e.printStackTrace();
    }
}
```

Este método recibe una nueva entidad y actualiza sus datos en el repositorio (update).

```java
@Override
public void actualizarCurso(Curso objCurso)
{
    Transaction transaction = null;
    try (Session session = HibernateUtil.getSessionFactory().openSession())
    {
        // start a transaction
        transaction = session.beginTransaction();

        // save the student object
        String hql = "UPDATE Curso SET nombre = :nombre "
                   + " , docente = :docente "
                   + " , lugar = :lugar "
                   + "WHERE idCurso = :ID";
        Query query = session.createQuery(hql);
        query.setParameter("nombre", objCurso.getNombre());
        query.setParameter("docente", objCurso.getDocente());
        query.setParameter("lugar", objCurso.getLugar());
        query.setParameter("ID", objCurso.getIdCurso());
        int result = query.executeUpdate();
        System.out.println("Rows affected: " + result);

        // commit transaction
        transaction.commit();
    }
```

```
100             catch (Exception e)
101             {
102                 if (transaction != null)
103                 {
104                     transaction.rollback();
105                 }
107                 e.printStackTrace();
107             }
108         }
```

Este método recibe una nueva entidad y elimina sus datos en el repositorio (delete).

```
109         @Override
            public void eliminarCurso(int id)
111         {
112
113             Transaction transaction = null;
114             try (Session session = HibernateUtil.getSessionFactory().openSession())
115             {
116                 // start a transaction
117                 transaction = session.beginTransaction();
118
119                 // Delete a student object
120                 Curso objCurso = session.get(Curso.class, id);
121                 if (objCurso != null)
122                 {
123                     String hql = "DELETE FROM Curso " + "WHERE idCurso = :ID";
124                     Query query = session.createQuery(hql);
125                     query.setParameter("ID", id);
126                     int result = query.executeUpdate();
127                     System.out.println("Rows affected: " + result);
128                 }
129
130                 // commit transaction
131                 transaction.commit();
132             }
133             catch (Exception e)
134             {
135                 if (transaction != null)
136                 {
137                     transaction.rollback();
138                 }
140                 e.printStackTrace();
140             }
141         }
```

Este método recibe un ID consulta al repositorio y devuelve los datos de la entidad. (select where).

```java
    @Override
    public Curso obtenerCurso(int id)
    {

        Transaction transaction = null;
        Curso objCurso = null;
        try (Session session = HibernateUtil.getSessionFactory().openSession())
        {
            // start a transaction
            transaction = session.beginTransaction();

            // get an student object
            String hql = " FROM Curso C WHERE C.id = :ID";
            Query query = session.createQuery(hql);
            query.setParameter("ID", id);
            List results = query.getResultList();

            if (results != null && !results.isEmpty())
            {
                objCurso = (Curso) results.get(0);
            }
            // commit transaction
            transaction.commit();
        }
        catch (Exception e)
        {
            if (transaction != null)
            {
                transaction.rollback();
            }
            e.printStackTrace();
        }
        return objCurso;
    }
```

Este método consulta al repositorio y devuelve una Lista de Entidades. (select all).

```java
    @Override
    public List<Curso> obtenerCursos()
    {
        try (Session session = HibernateUtil.getSessionFactory().openSession())
        {
            return session.createQuery("from Curso", Curso.class).list();
        }
        catch (Exception e)
        {
            e.printStackTrace();
            return null;
        }
    }
}

```

3.5 Crearemos el Servlet del **Controlador**.

En el paquete **Controlador** crearemos el **Servlet,** el cual realiza las funcionalidades del controlador, porque interactúa con las Vistas (páginas JSP) y las clases del Modelo (clases con hibernate).

```java
package Controlador;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import Modelo.dao.CursoDAO;
import Modelo.dao.ICursoDAO;
import Modelo.entity.Curso;
import java.util.List;
/**
 * Versión 1.0
 * @author Ing. Aníbal Sardón Paniagua
 */

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

    /** Processes requests for both HTTP <code>GET</code> and <code>POST</code> ...9 lines */
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {...15 lines }

    // <editor-fold defaultstate="collapsed" desc="HttpServlet methods. Click on the + sign on
    /** Handles the HTTP <code>GET</code> method ...8 lines */
```

El método GET es llamado desde los hipervínculos y los Formularios HTML

```java
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException
    {
        if( request.getParameter("metodo")!=null )
        {
            if( request.getParameter("metodo").equals("lista"))
            {   //CONSULTAR TODOS
                lista(request, response);
            }
            else
                if( request.getParameter("metodo").equals("modifica"))
                {   //CONSULTA UN CURSO PARA MODIFICAR
                    String id = request.getParameter("id");
                    CursoDAO objCursoDAO = new CursoDAO();
                    Curso objCurso = objCursoDAO.obtenerCurso(Integer.parseInt(id));
                    request.setAttribute("curso", objCurso);
                    request.getRequestDispatcher("/modificaCurso.jsp").forward(request, response);
                }
                else
                    if( request.getParameter("metodo").equals("elimina"))
                    {   //CONSULTA UN CURSO PARA ELIMINAR
                        String id = request.getParameter("id");
                        CursoDAO objCursoDAO = new CursoDAO();
                        Curso objCurso = objCursoDAO.obtenerCurso(Integer.parseInt(id));
                        request.setAttribute("curso", objCurso);
                        request.getRequestDispatcher("/eliminaCurso.jsp").forward(request, response)
                    }
        }
    }
```

El método POST es llamado desde los formularios HTML.

```
88
89      /** Handles the HTTP <code>POST</code> method ...8 lines */
97      @Override
        protected void doPost(HttpServletRequest request, HttpServletResponse response)
99              throws ServletException, IOException
100     {
101         if( request.getParameter("metodo")!=null )
102         {
                if( request.getParameter("metodo").equals("insertar"))
104             {
105                 registra(request, response);
106             }
107             else
108                 if( request.getParameter("metodo").equals("actualizar"))
109                 {
110                     actualiza(request, response);
111                 }
112                 else
113                     if( request.getParameter("metodo").equals("eliminar"))
114                     {
115                        elimina(request, response);
116                     }
117
118         }
119     }
```

Los siguientes métodos son llamados desde los métodos GET y POST del Servlet.

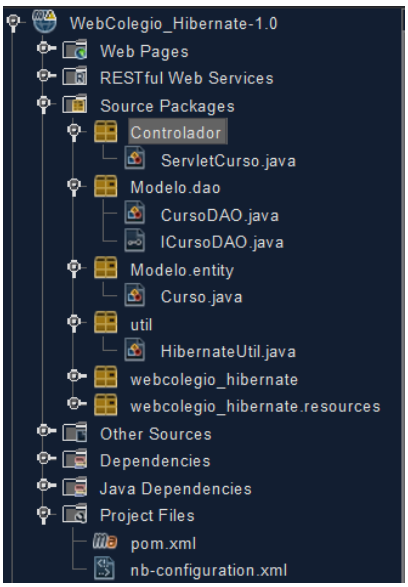
```
122     @Override
        public String getServletInfo() {
124         return "Short description";
125     }// </editor-fold>
126
127     protected void lista(HttpServletRequest request, HttpServletResponse response)
128             throws ServletException, IOException
129     {
130         ICursoDAO objCursoDAO = new CursoDAO();
131         // OBTENER TODOS
132         List<Curso> listaCursos = objCursoDAO.obtenerCursos();
133         //MUESTRA LA LISTA en JSP
134         request.setAttribute("cursos", listaCursos);
135         request.getRequestDispatcher("/listaCursos.jsp").forward(request, response);
136     }
```

```
137     protected void registra(HttpServletRequest request, HttpServletResponse response)
138             throws ServletException, IOException
139     {   //INSERTAR CURSO
140         String nombre = request.getParameter("nombre");
141         String docente = request.getParameter("docente");
142         String lugar = request.getParameter("lugar");
143
144         ICursoDAO objCursoDAO = new CursoDAO();
145         Curso objCurso = new Curso(0, nombre, docente, lugar);
146         objCursoDAO.grabarCurso(objCurso);
147
148         lista(request, response);
149     }
```

```java
    protected void actualiza(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException
    {   //MODIFICAR CURSO
        int id = Integer.parseInt( request.getParameter("id"));
        String nombre = request.getParameter("nombre");
        String docente = request.getParameter("docente");
        String lugar = request.getParameter("lugar");

        ICursoDAO objCursoDAO = new CursoDAO();
        Curso objCurso = new Curso(id, nombre, docente, lugar);
        objCursoDAO.actualizarCurso(objCurso);

        lista(request, response);
    }
```

```java
    protected void elimina(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException
    {   //ELIMINAR CURSO
        int id = Integer.parseInt( request.getParameter("id"));
        ICursoDAO objCursoDAO = new CursoDAO();
        objCursoDAO.eliminarCurso(id);

        lista(request, response);
    }
}
```

Hasta éste momento el proyecto tiene los siguientes paquetes y archivos creados:



4. Creación de las páginas JSP (Vista)

4.1 Página **index.html**

En la página index.html agregaremos las siguientes etiquetas HTML:

```
                    <div class="col-lg-12">
                        <div class="cont mb-5" style="width: 60%;">
                            <div class="head">
                                <h1>Bienvenido al Sitio WEB</h1> <br> <br>
                            </div>
                            <button type="button" class="btn-send" id="btnsend" onclick="">
                                <a href="ServletCurso?metodo=lista">Listar Cursos</a>
                            </button><br>
                            <button type="button" class="btn-send" id="btnsend" onclick="">
                                <a href="buscarCurso.jsp">Buscar Curso</a>
                            </button><br>
                        </div>
                    </div>
```

4.2 Crearemos la página **listarCursos.jsp**

La página JSP tendrá las siguientes etiquetas HTML y ScriptLets:

```
                    <div class="head">
                        <h1>Listado de Cursos</h1>
                    </div><br>
                    <h3>
                        <a href="insertaCurso.jsp">Registrar Curso</a>
                    </h3><br>
                <table>
                    <caption class=""> Lista de Cursos</caption>

                    <tr class="">
                      <th>Id</th><th>Nombre</th><th>Docente</th><th>Lugar</th><th></th><th></th>
                    </tr>
                    <!--
                      Scriplet: son inserciones de codigo java dentro un JSP    < % %>
                      Expression: son resultados de codigo java que se va visualizar en el JSP < %=   %>
                     -->
                <%
                    List<Curso> lista =(List<Curso>)request.getAttribute("cursos");
                    if(lista != null)
                    {
                        for(Curso aux :lista)
                        {
                %>
```

```
                        <tr class="fila_campo">
                                <td><%= aux.getIdCurso() %></td>
                                <td><%= aux.getNombre()  %></td>
                                <td><%= aux.getDocente()%></td>
                                <td><%= aux.getLugar()%></td>

                                <td>
                                    <a href="ServletCurso?metodo=elimina&id=<%= aux.getIdCurso()%>">
                                        <img alt="Elimina" src="images/Delete.gif">
                                    </a>
                                </td>
                                <td>
                                    <a href="ServletCurso?metodo=modifica&id=<%= aux.getIdCurso() %>">
                                        <img alt="Actualiza" src="images/Edit.gif">
                                    </a>
                                </td>
                        </tr>

                <%          }
                    }
                %>

                </table>
```

4.3 Crearemos la página **insertarCurso.jsp**

La página JSP tendrá las siguientes etiquetas HTML.

```
<div class="head">
    <h1>Registrar Curso</h1>
</div>
<form action="ServletCurso" method="post">
    <input type="hidden" name="metodo" value="insertar"/>
    <table class="table">
        <tr>
            <td class="">ID</td>
            <td class="">
                <input type="text" name="id"  readonly value=""></td>
        </tr>
        <tr>
            <td class="">Nombre del Curso</td>
            <td class="">
                <input type="text" name="nombre" required  value=""></td>
        </tr>
        <tr>
            <td class="">Docente</td>
            <td class="">
                <input type="text" name="docente" required value=""></td>
        </tr>
```

```
        <tr>
            <td class="">Lugar</td>
            <td class="">
                <input type="text" name="lugar" required value=""></td>
        </tr>
        <tr>
            <td class=""><input type="submit" value="enviar"></td>
            <td class=""><input type="reset" value="borrar"></td>
        </tr>
    </table>
</form>
</div>
```

4.4 Crearemos la página **modificarCurso.jsp**

La página JSP tendrá las siguientes etiquetas HTML y ScriptLets.

```
<h1 class="loginTitle">Información del Curso</h1>
<%   Curso objCurso = (Curso) request.getAttribute("curso"); %>
<form action="ServletCurso" method="post">
    <input type="hidden" name="metodo" value="actualizar"/>
    <table class="table">
        <tr>
            <td class="loginLabel">ID</td>
            <td class="loginControl">
                <input type="text" name="id"  readonly value="<%= objCurso.getIdCurso() %>"></td>
        </tr>
        <tr>
            <td class="loginLabel">Nombre del Curso</td>
            <td class="loginControl">
                <input type="text" name="nombre" required  value="<%= objCurso.getNombre() %>"></td>
        </tr>
        <tr>
            <td class="loginLabel">Docente</td>
            <td class="loginControl">
                <input type="text" name="docente" required value="<%= objCurso.getDocente() %>"></td>
        </tr>
```

```
        <tr>
            <td class="loginLabel">Lugar</td>
            <td class="loginControl">
                <input type="text" name="lugar" required value="<%= objCurso.getLugar() %>"></td>
        </tr>
        <tr>
            <td class="loginButton"><input type="submit" value="enviar"></td>
            <td class="loginButton"><input type="reset" value="borrar"></td>
        </tr>
    </table>
</form>
```

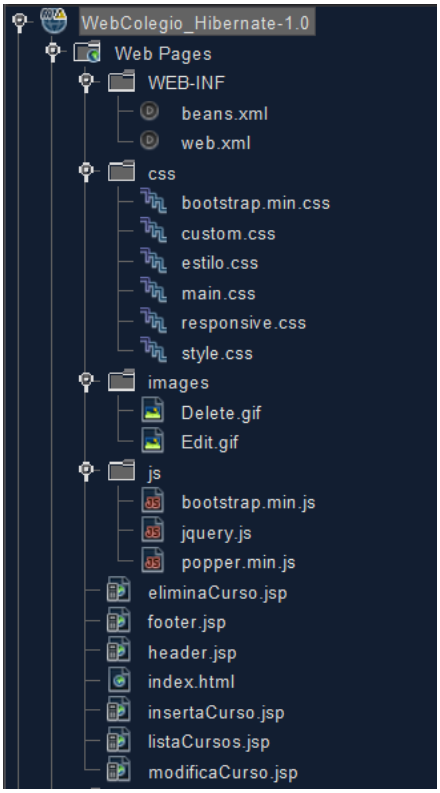
4.5 Crearemos la página **eliminaCurso.jsp**

La página JSP tendrá las siguientes etiquetas HTML y ScriptLets.

```
<body>
<h1 class="loginTitle">Eliminar Curso</h1>
<%   Curso objCurso = (Curso) request.getAttribute("curso"); %>
<form action="ServletCurso" method="post">
    <input type="hidden" name="metodo" value="eliminar"/>
    <table class="table">
        <tr>
            <td class="loginLabel">ID</td>
            <td class="loginControl">
                <input type="text" name="id"  readonly value="<%= objCurso.getIdCurso() %>"></td>
        </tr>
        <tr>
            <td class="loginLabel">Nombre del Curso</td>
            <td class="loginControl">
                <input type="text" name="nombre" required  value="<%= objCurso.getNombre() %>"></td>
        </tr>
        <tr>
            <td class="loginLabel">Docente</td>
            <td class="loginControl">
                <input type="text" name="docente" required value="<%= objCurso.getDocente() %>"></td>
        </tr>
```

```
        <tr>
            <td class="loginLabel">Lugar</td>
            <td class="loginControl">
                <input type="text" name="lugar" required value="<%= objCurso.getLugar() %>"></td>
        </tr>
        <tr>
            <td class="loginButton"><input type="submit" value="enviar"></td>
            <td class="loginButton"><input type="reset" value="borrar"></td>
        </tr>
    </table>
</form>
```

Al final el proyecto Web tendrá los siguientes paquetes, páginas JSP, páginas HTML y archivos recursos.



8. ENTREGABLES

Luego de levantar la aplicación MVC con Hibernate realice los siguientes entregables:

ENTREGABLES

1. Revisar y corregir el proyecto Creado para que cumpla con los principios **SOLID y COC.**

2. Crear las páginas JSP, Servlets, clases (patrón MVC) e Hibernate, para realizar las operaciones básicas con **tabla Cursos.**
   - La página **BuscarCurso.JSP** permite al usuario ingresar un **nombre de Curso** o el **Nombre del Docente** y buscar todas las coincidencias.
   - El sistema muestra una lista de Cursos con las coincidencias.
   - El usuario puede seleccionar uno de los cursos listados y el sistema le mostrará el formulario con los datos del curso para: *actualizar sus datos*, o *eliminarlo*.
   - Si el usuario quiere crear un *nuevo Curso*, Ingresa a la opción **"Nuevo Curso"**, el sistema le muestra un formulario vacío para que ingrese el ***nombre***, ***lugar*** y seleccione un docente de la ***Lista Desplegable*** (combobox) que muestra los nombres de la **Tabla docentes.**
   - Al finalizar el usuario presiona el *botón "Registrar"* para grabar los datos en el sistema.

3. Crear las páginas JSP, Servlets, clases (patrón MVC) e Hibernate, para realizar las operaciones básicas con **tabla Docentes.**

   [ID | Nombres | Apellidos | Especialidad | FechaNacimiento]

   - Buscar por Nombres o Por Apellidos o Por Fecha de Nacimiento -> Rango de Fechas (Desde: - Hasta:)
   - Agregar un Nuevo Docente
   - Modificar los Datos del Docente
   - Eliminar el Docente

   NOTA:
   Utilizar la arquitectura MVC con Hibernate planteada en el presente ejercicio.
   Los elementos diseñados y construidos deben cumplir con los **principios SOLID**.

9. FUENTES DE INFORMACIÓN COMPLEMENTARIA

- Cervantes, H & Velasco,P (2016). Arquitectura de Software, Conceptos y ciclo de Desarrollo. Mexico D.F. Cengage Learning Editores.
- Somerville, I.(2015). Ingeniería de Software. Madrid, España. Pearson. 7ma. Edición.
- Pressman, R.(2015). Ingeniería de Software, un enfoque práctico. Mexico DF. Mc Graw Hill. 7ma. Edición.