	<p style="text-align: center;"><b>UNIVERSIDAD DON BOSCO</b> <b>ESCUELA DE COMPUTACION</b></p>
<p style="text-align: center;"><b>Ciclo II</b></p>	<p style="text-align: center;"><b>PROGRAMACION ORIENTADA A OBJETOS</b> <b>INTRODUCCION AL MODELO DE SARROLLO MVC</b></p>

## I. OBJETIVOS.

- Que el alumno comprenda el patrón de diseño MVC y pueda implementarlo en una aplicación web.

## II. INTRODUCCIÓN.

Modelo Vista Controlador (MVC) es un estilo de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos.

Se trata de un modelo muy maduro y que ha demostrado su validez a lo largo de los años en todo tipo de aplicaciones, y sobre multitud de lenguajes y plataformas de desarrollo.

- El **Modelo** que contiene una representación de los datos que maneja el sistema, su lógica de negocio, y sus mecanismos de persistencia.
- La **Vista**, o interfaz de usuario, que compone la información que se envía al cliente y los mecanismos interacción con éste.
- El **Controlador**, que actúa como intermediario entre el Modelo y la Vista, gestionando el flujo de información entre ellos y las transformaciones para adaptar los datos a las necesidades de cada uno.

El modelo es el responsable de:

- Acceder a la capa de almacenamiento de datos. Lo ideal es que el modelo sea independiente del sistema de almacenamiento.
- Define las reglas de negocio (la funcionalidad del sistema). Un ejemplo de regla puede ser: "Si la mercancía pedida no está en el almacén, consultar el tiempo de entrega estándar del proveedor".
- Lleva un registro de las vistas y controladores del sistema.
- Si estamos ante un modelo activo, notificará a las vistas los cambios que en los datos pueda producir un agente externo (por ejemplo, un fichero por lotes que actualiza los datos, un temporizador que desencadena una inserción, etc.).

El controlador es responsable de:

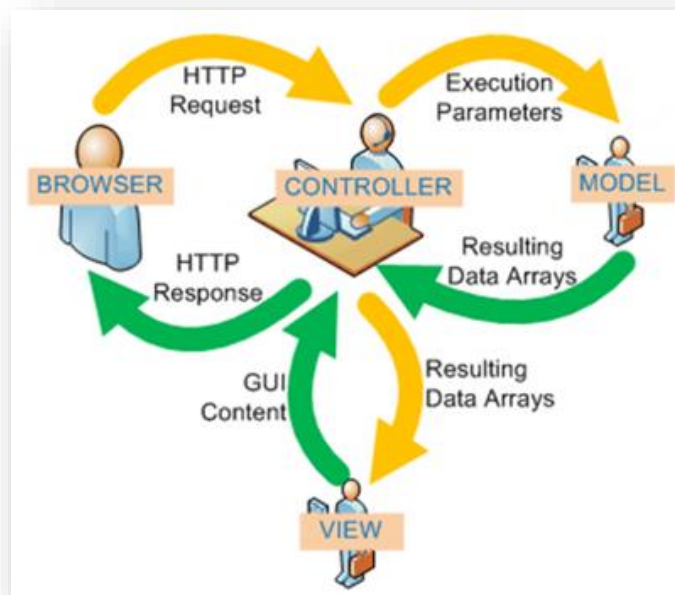
- Recibe los eventos de entrada (un clic, un cambio en un campo de texto, etc.).
- Contiene reglas de gestión de eventos, del tipo "SI Evento Z, entonces Acción W". Estas acciones pueden suponer peticiones al modelo o a las vistas. Una de estas peticiones a las

vistas puede ser una llamada al método "Actualizar()". Una petición al modelo puede ser "Obtener\_tiempo\_de\_entrega ( nueva\_orden\_de\_venta )".

Las vistas son responsables de:

- Recibir datos del modelo y los muestra al usuario.
- Tienen un registro de su controlador asociado (normalmente porque además lo instancia).

El flujo que sigue el control generalmente es el siguiente:



El usuario interactúa con la interfaz de usuario de alguna forma (por ejemplo, el usuario pulsa un botón, enlace, etc.)

1. El usuario interactúa con la interfaz de usuario de alguna forma (por ejemplo, el usuario pulsa un botón, enlace, etc.)
2. El controlador recibe (por parte de los objetos de la interfaz-vista) la notificación de la acción solicitada por el usuario. El controlador gestiona el evento que llega, frecuentemente a través de un gestor de eventos (handler) o callback.
3. El controlador accede al modelo, actualizándolo, posiblemente modificándolo de forma adecuada a la acción solicitada por el usuario (por ejemplo, el controlador actualiza el carro de la compra del usuario). Los controladores complejos están a menudo estructurados usando un patrón de comando que encapsula las acciones y simplifica su extensión.
4. El controlador delega a los objetos de la vista la tarea de desplegar la interfaz de usuario. La vista obtiene sus datos del modelo para generar la interfaz apropiada para el usuario donde se refleja los cambios en el modelo (por ejemplo, produce un listado del contenido

del carro de la compra). El modelo no debe tener conocimiento directo sobre la vista. Sin embargo, se podría utilizar el patrón Observador para proveer cierta indirección entre el modelo y la vista, permitiendo al modelo notificar a los interesados de cualquier cambio. Un objeto vista puede registrarse con el modelo y esperar a los cambios, pero aun así el modelo en sí mismo sigue sin saber nada de la vista. El controlador no pasa objetos de dominio (el modelo) a la vista aunque puede dar la orden a la vista para que se actualice. Nota: En algunas implementaciones la vista no tiene acceso directo al modelo, dejando que el controlador envíe los datos del modelo a la vista.

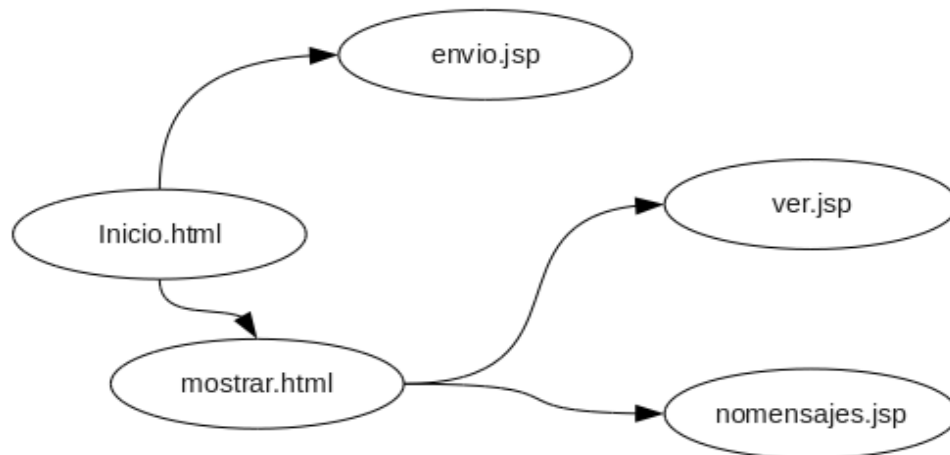
5. La interfaz de usuario espera nuevas interacciones del usuario, comenzando el ciclo nuevamente.

### III. PROCEDIMIENTO.

#### Descripción

Para comprender la importancia de esta arquitectura, vamos a desarrollar una aplicación Web siguiendo este patrón MVC. La aplicación consistirá en un sencillo sistema de envío y visualización de mensajes a través de la Web, cuyas páginas se muestran más adelante. Cada mensaje estará formado por un destinatario, remitente y un texto.

La página de dos enlaces con las opciones del usuario, la de visualización de mensajes le llevará a otra página (“mostrar.jsp”) donde se le solicitará el nombre del destinatario cuyos mensajes quiere visualizar. En caso de tener mensajes asociados se le enviará a una página donde se le mostrará una tabla con todos sus mensajes, indicando para cada uno de ellos el remitente y el contenido del mensaje. Por otro lado, la opción de envío de mensajes le llevará a una página en la que se le solicitarán los datos del mensaje que quiere enviar, devolviéndolo después a la página de inicio de una vez que el mensaje ha sido almacenado.



#### Desarrollo

Los mensajes manejados por la aplicación serán almacenados en una base de datos llamada “**mensajes**” que contendrá una tabla también llamada “**mensajes**” y que contendrá los siguientes campos.

Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	NULL	auto_increment
remitente	varchar(60)	NO		NULL	
destinatario	varchar(60)	NO		NULL	
texto	varchar(200)	NO		NULL	

## Desarrollo

Creación de la aplicación utilizando el modelo MVC.

El desarrollo de esta aplicación se realizará siguiendo el patrón Modelo Vista Controlador, donde tendremos un servlet llamado “**Controlador**” en el que se centralizarán todas las peticiones procedentes desde el cliente.

El Modelo estará implementado mediante una clase a la que llamaremos **Operaciones** que dispondrá de dos métodos: **grabarMensaje()**, encargado de almacenar en la base de datos los datos de un mensaje, y **obtenerMensajes()**, cuya función será la de recuperar la lista de mensajes asociados al destinatario que se proporciona como parámetro.

Los mensajes serán manipulados mediante una clase JavaBean llamada **Mensaje**, que encapsulará los tres datos asociados a un determinado mensaje.

En cuanto a las vistas, serán implementadas mediante cinco páginas, dos HTML (inicio.html, mostrar.html) y tres JSP (envio.jsp, ver.jsp, nomensajes.jsp). Utilizando el parámetro “**operacion**” insertado en la URL, las páginas inicio.html, mostrar.html y envio.jsp indicarán al servlet controlador el tipo de acción que se debe llevar a cabo en cada petición.

A continuación, proceder a crear las clases, páginas jsp y html con eclipse, para ello seguir los siguientes pasos:

1. Crear el proyecto web llama **MvcIntro**.
2. Agregar los objetos de la complementarios proporcionados como son **css**, **js** y las páginas **error404.jsp** y **error500.jsp** al proyecto creado.

### error404.jsp

```
<%--
Created by IntelliJ IDEA.
User: emers
Date: 4/18/2022
Time: 8:30 PM
To change this template use File | Settings | File Templates.
--%>
```

```
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>
  <title>Error 404</title>
</head>
<body>
<h1>Error 404</h1>
</body>
</html>
```

### error500.jsp

```
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>
  <title>Error 500</title>
</head>
<body>
<h1>Error 500</h1>
</body>
</html>
```

**NOTA:** No olvidar marcar los recursos css y js como **resources root**, de no recordarlo consultar las guías anteriores.

3. Agregar las dependencias necesarias dentro del archivo pom.xml, para este caso usaremos el mysqlconnector y jstl. Se le proporcionan en este paso. De no recordar como agregarlas a este proyecto, consultar los materiales anteriores.

```
<!--      https://mvnrepository.com/artifact/org.glassfish.web/jakarta.servlet.jsp.jstl      -->
<dependency>
  <groupId>org.glassfish.web</groupId>
  <artifactId>jakarta.servlet.jsp.jstl</artifactId>
  <version>2.0.0</version>
</dependency>
<!--      https://mvnrepository.com/artifact/mysql/mysql-connector-java      -->
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>8.0.27</version>
</dependency>
```

3. Crear la clase “**MensajeBean**” dentro del paquete “**javabeans**” y agregar el siguiente código.

```

package javabeans;

public class MensajeBean {
    private String remite;
    private String destino;
    private String texto;

    public MensajeBean() { }

    public MensajeBean(String remite, String destino, String texto)
    {
        this.remite = remite;
        this.destino = destino;
        this.texto = texto;
    }
    public String getRemite() {
        return remite;
    }
    public void setRemite(String remite) {
        this.remite = remite;
    }
    public String getDestino() {
        return destino;
    }
    public void setDestino(String destino) {
        this.destino = destino;
    }
    public String getTexto() {
        return texto;
    }
    public void setTexto(String texto) {
        this.texto = texto;
    }
}

```

4. Como siguiente punto crearemos el modelo, para ello crear una clase llamada “Operaciones” en un paquete llamado “modelo”

```

package modelo;
import java.sql.*;
import javabeans.*;

import java.util.*;

```

```

public class Operaciones {
    public Connection getConnection(){
        Connection cn=null;
        try{
            Class.forName("com.mysql.jdbc.Driver");

            // Se obtiene una conexión con la base de datos. Cambie los parámetros por sus datos
            cn = DriverManager.getConnection (
                "jdbc:mysql://localhost/mensajes","root", "");
        }
        catch(Exception e){e.printStackTrace();}

        return cn;
    }

    public ArrayList<MensajeBean> obtenerMensajes(String destino){
        Connection cn=null;
        ArrayList<MensajeBean> mensaje=null;
        Statement st;
        ResultSet rs;
        try {
            cn=getConnection();
            st=cn.createStatement();
            String tsq;
            tsq="select * from mensajes where destinatario='"+destino+"'";
            rs=st.executeQuery(tsq);
            mensaje=new ArrayList<MensajeBean>();
            while (rs.next()) {
                MensajeBean m=new
MensajeBean(rs.getString("remite"nte"),rs.getString("destinatario"),
                rs.getString("texto"));
                mensaje.add(m);
            }
            cn.close();

        } catch (Exception e) {
            e.printStackTrace();
        }
        return(mensaje);
    }

    public boolean grabaMensaje(MensajeBean m){
        Connection cn;
        Statement st;
        ResultSet rs;
    }

```

```

        boolean result = false;
        try {
            cn=getConnection();
            st=cn.createStatement();
            String tsq;
            tsq="Insert into mensajes values(null,'";tsq+=m.getRemite()+"',"+ m.getDestino()+"',"+
m.getTexto()+"')";
            st.execute(tsq);
            cn.close();
            result = true;
        } catch (Exception e) {
            result = false;
            e.printStackTrace();
        }
        return result;
    }
}

```

5. Crear el controlador como ya se había mencionado será un **servlet** que será llamado **“Controlador”** que estará dentro de un paquete llamado **“servlets”**.

```

package servlets;

import jakarta.servlet.*;
import jakarta.servlet.http.*;
import jakarta.servlet.annotation.*;
import javabeans.MensajeBean;

import modelo.*;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;

@WebServlet(name = "Controlador", value = "/Controlador")
public class Controlador extends HttpServlet {

    protected void processRequest (HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {
            String op=request.getParameter("operacion");
            int result=0;
            if(op.equals("envio")){
                response.sendRedirect("envio.jsp");
            }
        }
    }
}

```



```

    }else if(op.equals("grabar")){
        MensajeBean men= (MensajeBean) request.getAttribute("mensaje");
        Operaciones oper= new Operaciones();

        if (oper.grabaMensaje(men)){
            request.setAttribute("result","ok");
        }
        RequestDispatcher rd=request.getRequestDispatcher("/inicio.jsp");
        rd.forward(request, response);
    }
    else if(op.equals("muestra")){
        response.sendRedirect("mostrar.jsp");
    }
    else if(op.equals("ver")){
        Operaciones oper=new Operaciones();
        ArrayList mensajes=oper.obtenerMensajes(request.getParameter("nombre"));
        request.setAttribute("mensajes", mensajes);
        RequestDispatcher rd=request.getRequestDispatcher("/ver.jsp");
        rd.forward(request, response);
    }

}
finally {
    out.close();
}

}
@Override
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    processRequest(request,response);
}

@Override
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    processRequest(request,response);
}
}

```

6. Configurar el archivo **web.xml** tal y como se muestra a continuación.

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="https://jakarta.ee/xml/ns/jakartaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

```

```
xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
https://jakarta.ee/xml/ns/jakartaee/web-app_5_0.xsd"
version="5.0">
<session-config>
  <session-timeout>
    30
  </session-timeout>
</session-config>

<context-param>
  <param-name>jakarta.servlet.jsp.jstl.fmt.localizationContext</param-name>
  <param-value>AplicationResource</param-value>
</context-param>

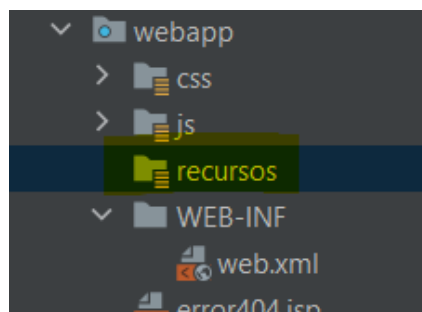
<error-page>
  <error-code>404</error-code>
  <location>/error404.jsp</location>
</error-page>

<error-page>
  <error-code>500</error-code>
  <location>/error500.jsp</location>
</error-page>

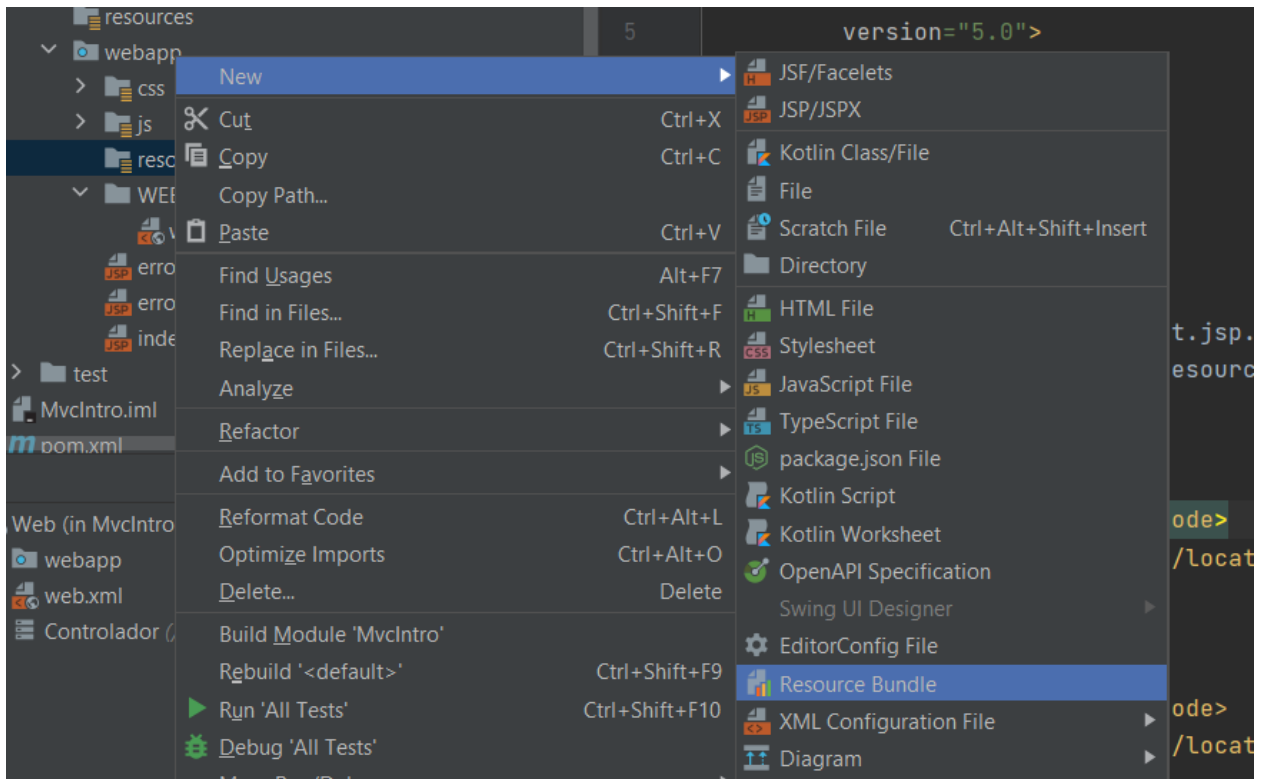
</web-app>
```

## 7. Uso de la internacionalización.

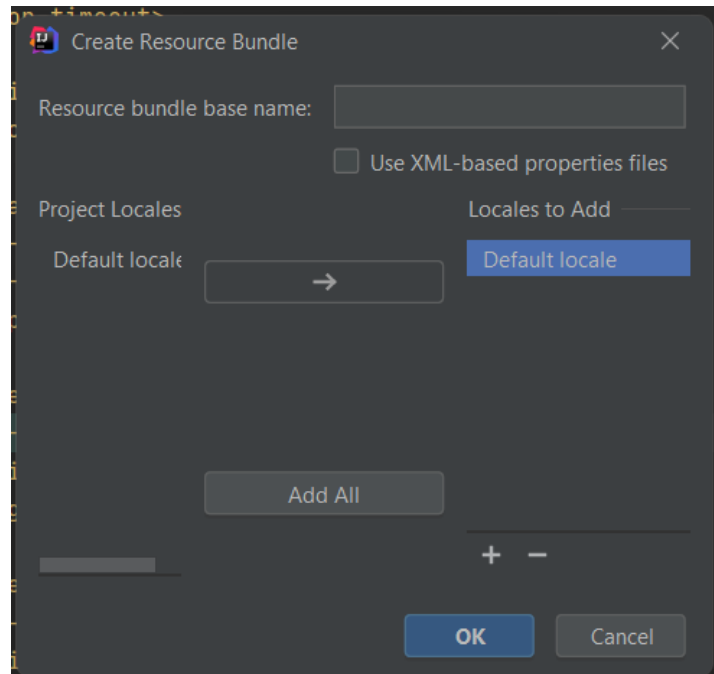
- Crear un nuevo directorio dentro de la carpeta **webapp**, este nuevo directorio se llamará **recursos**. No olvidar marcar esta carpeta como un **resources root**.



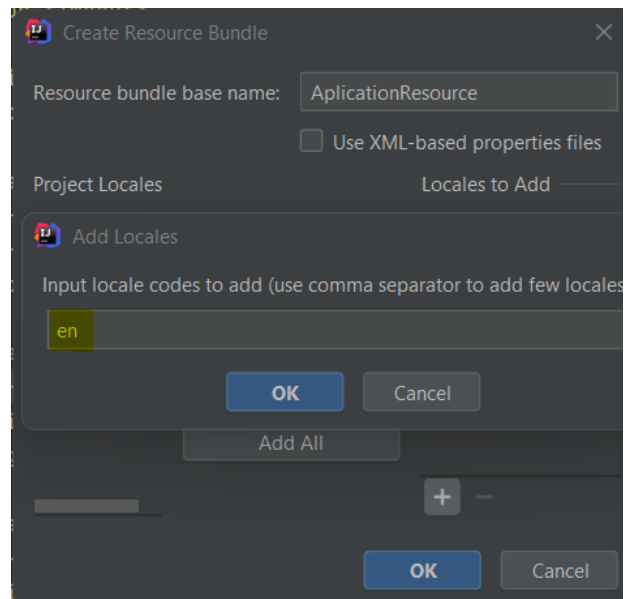
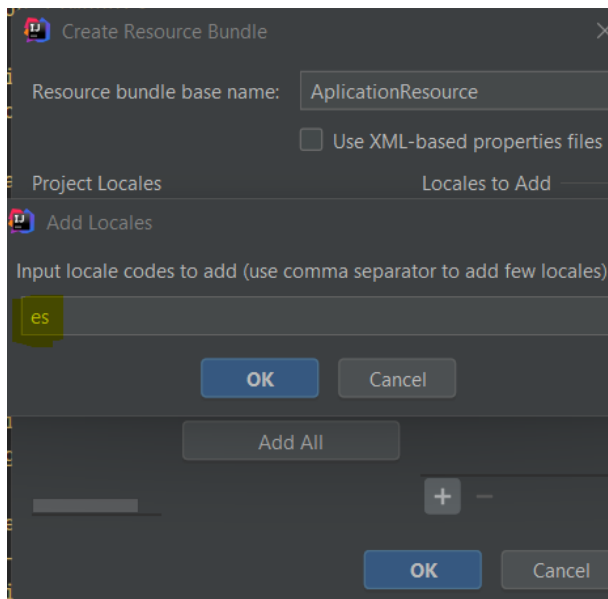
- Dar click derecho sobre esa carpeta New > Resource Bundle



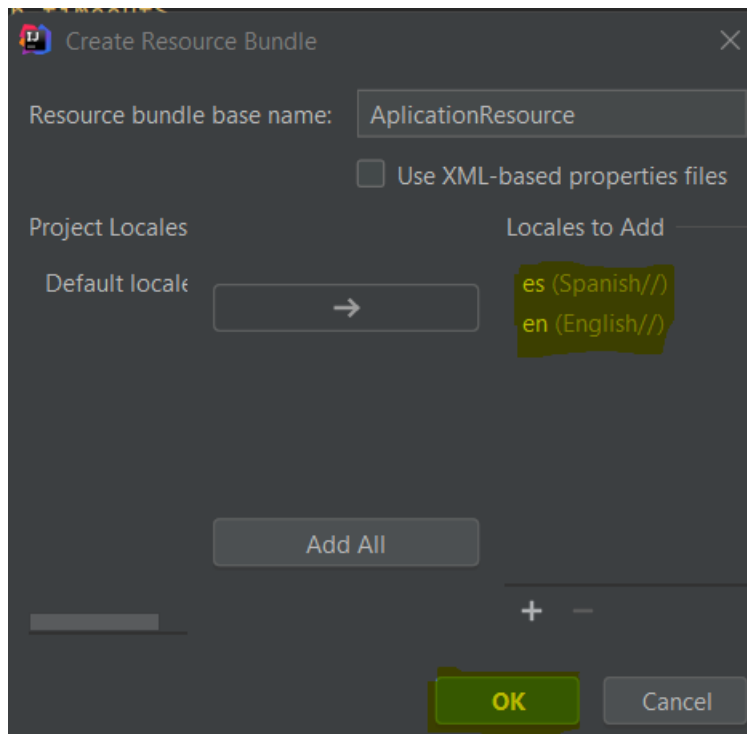
- Por el momento solo manejaremos 2 configuraciones para el lenguaje, para ello seleccionar **default locale** y dar click en el signo menos (-).



- Ahora asignaremos un nombre al recurso el cual será **AplicationResource**. Finalmente agregaremos los lenguajes correspondientes, en este caso usaremos el diminutivo **es** (para el lenguaje español) y **en** (para el lenguaje ingles).



- Finalmente deberá verse así y dará click en ok.



- Procederemos a agregar las llaves de identificación en sus archivos AplicationResource respectivos.

Español	Ingles
enviar=Enviar Mensaje	enviar=Send message
leer=Leer Mensaje	leer=Read message
generacion=Generación de Mensajes	generacion=Create Message
datosmensaje=Datos del Mensaje:	datosmensaje=Message Data
destinatario=Introduzca destinatario:	destinatario=Enter Recipient
remitente=Introduzca remitente:	remitente=Enter Sender
texto=Introduzca texto:	texto=Enter Text
enviar=Enviar	enviar=Send
limpiar=Limpiar	limpiar=Clean
mensajepara=Mensaje Para	mensajepara=Message For
remitente=Remitente:	remitente=Sender:
mensaje=Mensaje	mensaje=Message
nohaymensaje=No hay mensajes	nohaymensaje=No messages
introducirnombre=Introduzca su nombre:	introducirnombre=Enter your name
nomensajes=No hay mensajes para:	nomensajes=There are no message for:
inicio=Inicio	inicio=Start
paginaver=Ver Mensajes	paginaver=See Messages
nohaymensajes=No hay Mensajes	nohaymensajes=No messages
enviarmensaje=Enviar Mensajes	enviarmensaje=Send Messages
exitoso=Mensaje Ingresado Exitosamente	exitoso=Message enter successfully

8. Agregar el siguiente código a la página **inicio.jsp**

```

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<!DOCTYPE html>
<html>
<head>
    <title><fmt:message key="generacion"/></title>
    <link href="css/tabla1.css" rel="stylesheet" type="text/css" />
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="css/bootstrap.min.css">
</head>
<body>
<div class="container ">
    <div class="row">
        <div class="col-sm-4 col-sm-offset-4">
            <br/><br/>
            <a href="Controlador?operacion=envio" class="btn btn-info"><fmt:message
key="enviar"/></a>
            <br/><br/>
            <a href="Controlador?operacion=muestra" class="btn btn-success"><fmt:message
key="leer"/></a>

            <c:if test="\${not empty result}">
                <div class="alert alert-info">
                    <strong>Mensaje!</strong> <fmt:message key="exitoso"/>
                    <br>
                </div>
            </c:if>
        </div>
    </div>
</div>
</body>
</html>

```

9. Crear una página jsp llamada **mostrar.jsp** y agregar el siguiente código.

```

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<!DOCTYPE html>
<html>
<head>
    <link href="css/tabla1.css" rel="stylesheet" type="text/css" />
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" href="css/bootstrap.min.css">
    <title><fmt:message key="label.paginaver"/></title>

```

```

</head>
<body>
<div class="container">
  <div class="row">
    <div class="col-sm-4 col-sm-offset-4">
      <form action="Controlador?operacion=ver" method="post">

        <div class="form-group">
          <label for="nombre"><fmt:message key="introducirnombre" /></label>
          <input class="form-control" type="text" name="nombre" id="nombre">
        </div>
        <input class="btn btn-success" type="submit">
      </form>
    </div>
  </div>
</div>
</body>
</html>

```

10. Proceder a crear las páginas JSP llamada **envio.jsp** y agregar el siguiente código.

```

<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html>
<html>
<head>
  <link href="css/tabla1.css" rel="stylesheet" type="text/css" />
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="css/bootstrap.min.css">
  <title><fmt:message key="enviarmensaje"/></title>
</head>
<jsp:useBean id="mensaje" scope="request" class="javabeans.MensajeBean" />
<jsp:setProperty name="mensaje" property="*" />
<c:if test="${not empty param.texto}">
  <jsp:forward page="Controlador?operacion=grabar"/>
</c:if>

<body>
<div class="container">
  <div class="row">
    <div class="col-sm-4 col-sm-offset-4">
      <div class="row">
        <h1><fmt:message key="generacion"/></h1>
      </div>

      <form method="post">

```

```

<p class="center-block"><fmt:message key="datosmensaje" /></p>
<div class="form-group">
  <fmt:message key="destinatario"/>
  <input class="form-control" type="text" name="destino" >
</div>

<div class="form-group">
  <fmt:message key="remitente"/>
  <input type="text" class="form-control" name="remite">
</div>

<div class="form-group">
  <fmt:message key="texto"/>
  <textarea name="texto" class="form-control" id="texto"></textarea>
</div>
<hr><br/>
<input type="submit" class="btn btn-info" name="submit" value="<fmt:message
key="enviar"/>">
  <input type="reset" class="btn btn-info" value="<fmt:message key="limpiar"/>">
</form>
</div>
</div>
</body>
</html>

```

11. Proceder a crear las páginas JSP llamada **ver.jsp** y agregar el siguiente código.

```

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<!DOCTYPE html>
<html>
<head>
  <link href="css/tabla1.css" rel="stylesheet" type="text/css" />
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="css/bootstrap.min.css">
  <title><fmt:message key="paginaver"/></title>
</head>
<body>
<div class="container ">
  <div class="row">
    <div class="col-sm-4 col-sm-offset-4 ">
      <h1><fmt:message key="mensaje para"/> <c:out value="{param.nombre}"/> </h1>
      <table class="table table-hover table-bordered">

```



```

        <tr>
            <th><fmt:message key="remitente"/></th>
            <th><fmt:message key="mensaje"/></th>
        </tr>
        <c:forEach var="m" items="{mensajes}">
            <tr>
                <td><c:out value="{m.getRemite()}" /></td>
                <td><c:out value="{m.getTexto()}" /></td>
            </tr>
        </c:forEach>
    </table>
    <c:if test="{empty mensajes}">
        <fmt:message key="nohaymensaje" />

    </c:if>
    <br/>
    <a href="inicio.jsp" class="btn btn-info"><fmt:message key="inicio" /></a>
</div>
</div>
</body>
</html>

```

12. Proceder a crear las páginas JSP llamada **nomensajes.jsp** y agregar el siguiente código.

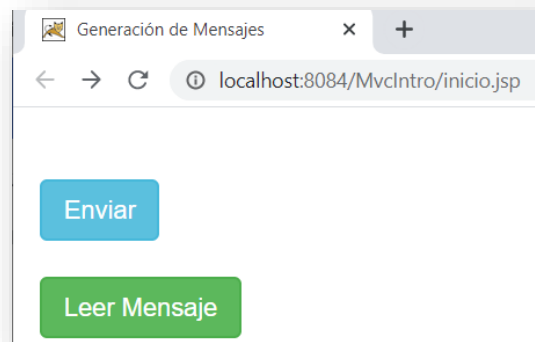
```

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html>
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title><fmt:message key="nohaymensajes" /></title>
</head>
<body>
<center>
    <h2><fmt:message key="nomensajes" /> <c:out value="{nombre}" /> </h2>
    <br><br><br>
    <a href="inicio.jsp"><fmt:message key="inicio" /></a>
</center>
</body>
</html>

```

13. Ejecutar la página llamada **inicio.jsp** y se deben de ver los siguientes resultados.

Página de inicio.



Página de envío de mensaje.

## Generación de Mensajes

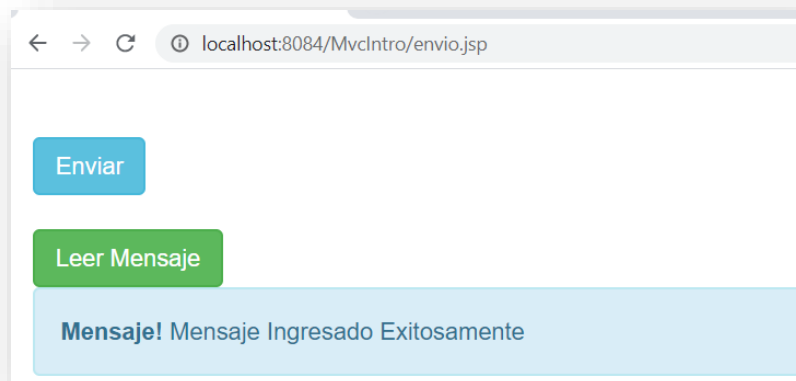
Datos del Mensaje:

Introduzca destinatario:

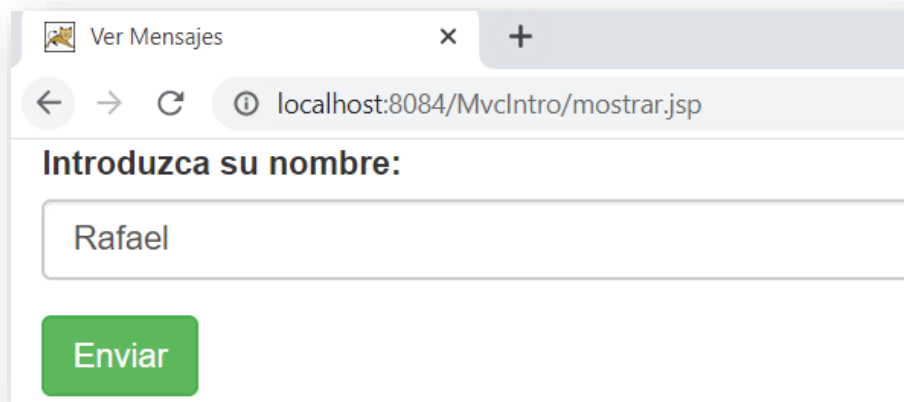
Remitente:

Introduzca texto:

Resultado después de enviar el Mensaje.

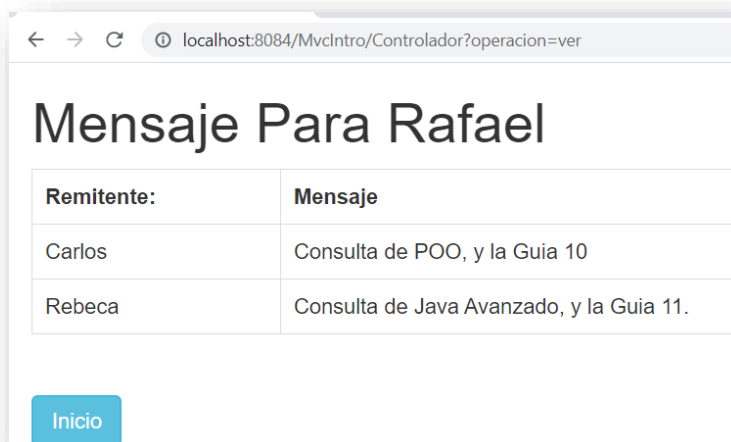


Lectura de Mensajes de Enviados a un usuario.



A screenshot of a web browser window with the title "Ver Mensajes". The address bar shows "localhost:8084/MvcIntro/mostrar.jsp". The page content includes the text "Introduzca su nombre:" followed by a text input field containing "Rafael". Below the input field is a green button labeled "Enviar".

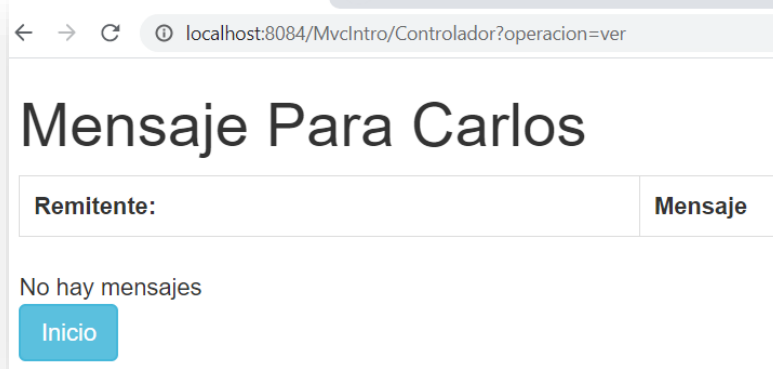
Resultados de los mensajes consultados.



A screenshot of a web browser window showing the results of a message query. The address bar shows "localhost:8084/MvcIntro/Controlador?operacion=ver". The page title is "Mensaje Para Rafael". Below the title is a table with two columns: "Remitente:" and "Mensaje". The table contains two rows of data. At the bottom of the page is a blue button labeled "Inicio".

Remitente:	Mensaje
Carlos	Consulta de POO, y la Guia 10
Rebeca	Consulta de Java Avanzado, y la Guia 11.

Resultado cuando no hay mensajes a mostrar.



## EJERCICIOS COMPLEMENTARIOS

Utilizando el patrón de diseño mvc cree la base de datos denominada **“inventario”** y crear operaciones CRUD **sobre la tabla ventas**.

```
CREATE TABLE IF NOT EXISTS `lineas_de_venta` (  
  `id_linea` int(4) NOT NULL AUTO_INCREMENT,  
  `Linea` varchar(30) NOT NULL,  
  PRIMARY KEY (`id_linea`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=5 ;
```

```
INSERT INTO `lineas_de_venta` (`id_linea`, `Linea`) VALUES  
(1, 'Linea Blanca'),  
(2, 'Electronica'),  
(3, 'Ferreteria'),  
(4, 'Hogar');
```

```
CREATE TABLE IF NOT EXISTS `ventas` (  
  `id_venta` int(4) NOT NULL AUTO_INCREMENT,  
  `id_linea` int(4) NOT NULL,  
  `fecha_venta` date NOT NULL,  
  `descripcion` varchar(50) NOT NULL,  
  PRIMARY KEY (`id_venta`),  
  KEY `id_linea` (`id_linea`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=9 ;
```

```
ALTER TABLE `ventas`
```

```
ADD CONSTRAINT `ventas_ibfk_1` FOREIGN KEY (`id_linea`) REFERENCES  
`lineas_de_venta` (`id_linea`) ON DELETE CASCADE ON UPDATE CASCADE;
```

## V. REFERENCIA BIBLIOGRAFICA

- Libros en biblioteca UDB.
  - **Struts** (Capítulo 1)
    - Antonio J. Martín Sierra
- <https://si.ua.es/es/documentacion/asp-net-mvc-3/1-dia/modelo-vista-controlador-mvc.html>  
Última fecha de visita: 19.04.2020