



UAEM

Universidad Autónoma
del Estado de México

Universidad Autónoma del Estado de México

Diplomado Superior en Desarrollo de
Software Empresarial

Alumnos: Benjamín Carpio Ramírez, Antonio Altamirano
Contreras, Itzel Hernández Valdivia

Módulo 7 Tecnologías del lado del servidor

Documentación Técnica “Control Feeding”

Docente: José Jair Vázquez Palma

ÍNDICE

ÍNDICE	1
PRESENTACIÓN	2
INTRODUCCIÓN	2
Funcionalidad general:.....	2
Descripción de funciones:.....	2
Descripción de especificaciones adicionales:	3
DIAGRAMAS DE UML	4
Diagrama de clases:	4
Diagrama de secuencia:	5
Diagrama de clases:	5
Diagrama modelo relacional:	6
Diccionario de datos:.....	7
ESTRUCTURA DEL PROYECTO CREADO EN IDE.....	8
Esquema (imagen) del proyecto creado en NetBeans.....	8
Descripción de cada una de las capas del patrón MVC.....	9
INTEGRACIÓN DEL PROYECTO WEB EN SERVIDOR.....	24
Descripción de cada paso y/o configuraciones para el alojamiento del proyecto en servidor.	24
Configuración del Servidor de Aplicaciones	27
Imágenes ilustrativas del correcto funcionamiento del sistema web en servidor.....	29
FUENTES CONSULTADAS.....	33

PRESENTACIÓN

En el ámbito deportivo, la alimentación es un pilar fundamental para alcanzar el máximo rendimiento. Una dieta bien estructurada puede marcar la diferencia entre el éxito y el fracaso. Es en este contexto donde nace Control Feding (traducido como *Alimentación Controlada*), una innovadora aplicación web diseñada por los desarrolladores Benjamín Carpio, Antonio Altamirano e Itzel Hernández, con el objetivo de asistir a atletas y entrenadores en la planificación y seguimiento de dietas personalizadas.

INTRODUCCIÓN

Funcionalidad general:

Control Feding es un sistema de consulta en línea para gestionar y optimizar los planes alimenticios para deportistas de alto rendimiento que componen BeautssFit. La aplicación ofrece un acceso fácil y rápido a la información nutricional necesaria para maximizar el potencial de cada uno de nuestros atletas, alineando la ingesta de alimentos con sus objetivos deportivos y necesidades individuales. Dentro de las funciones principales de dicha aplicación se tiene: la consulta de dietas personalizadas, visualización detallada de comidas, actualizaciones y ajustes.

Con ello se obtienen grandes beneficios como: la eficiencia y comodidad, optimización del rendimiento deportivo y la personalización.

Descripción de funciones:

Detalle de Atleta	
Descripción	Se da de alta a cada uno de los atletas ingresando su información general.
Datos:	Nombre
	Edad
	Peso
	Altura

Detalle de Dieta para Atleta	
Descripción	Se da de alta una comida con descripción y hora de su consumo.
Datos:	Comida
	Hora

Descripción de especificaciones adicionales:

Volver a la lista	
Descripción	Se puede visualizar la información de los atletas previamente registrados
	Id
	Nombre
	Edad
	Peso
	Altura

Guardar información	
Descripción	Se guarde la información registrada en los siguientes apartados. Esta información es almacenada en la base de datos.
	Detalle Atleta
	Detalle de la Dieta

Editar información	
Descripción	Se puede modificar la información registrada en cuanto a los campos:
	Comida
	Hora

Mostrar Dieta

Descripción	Se puede visualizar la información de las comidas registradas
	Id
	Comida
	Hora

DIAGRAMAS DE UML

Diagrama de clases:

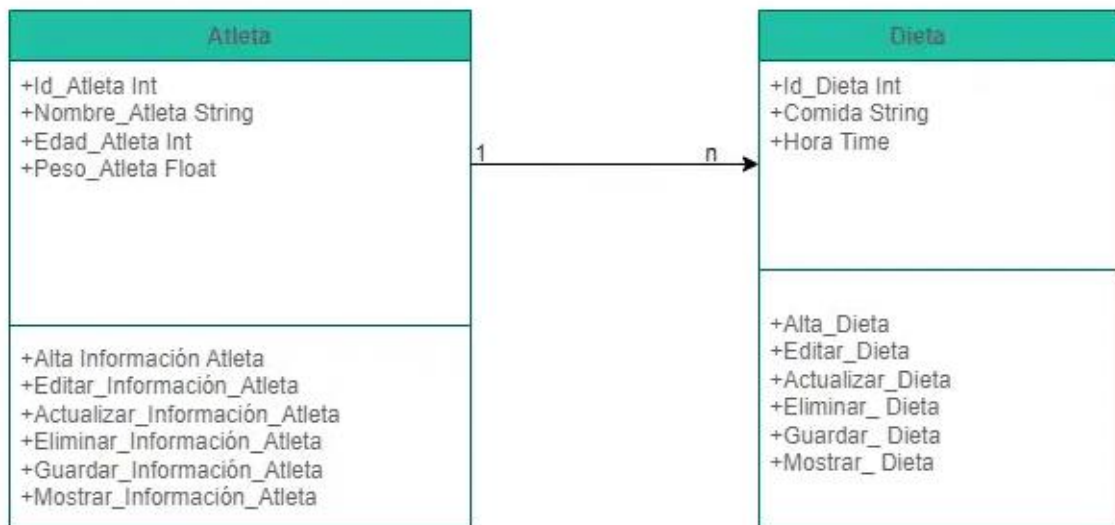


Tabla atletas

- Esta tabla almacena la información de los atletas que están siendo gestionados por el sistema.
- La tabla es utilizada para mostrar la lista de atletas, así como para crear, editar, y eliminar registros de atletas en el sistema. Cada atleta registrado aquí puede estar vinculado a una o varias dietas.

Tabla dietas

- Dicha tabla almacena las dietas asociadas a cada atleta. Cada dieta está compuesta por múltiples registros que detallan las comidas y suplementos que el atleta debe consumir, junto con los horarios correspondientes.
- La tabla permite gestionar las dietas de los atletas. Cada vez que se visualiza, edita o elimina una dieta, esta tabla es consultada. También se utiliza cuando se quiere añadir una nueva dieta a un atleta existente.

Diagrama de secuencia:

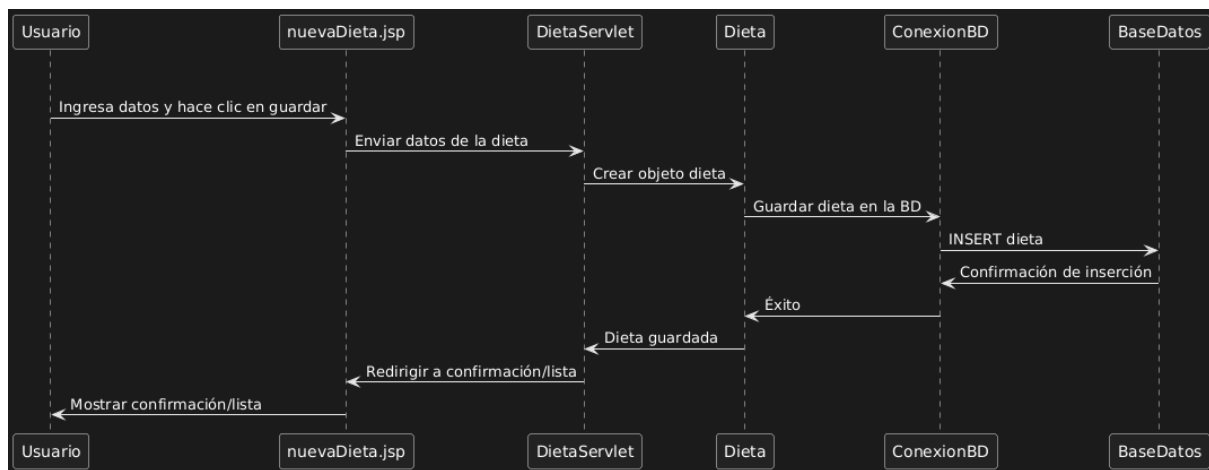


Diagrama de clases:

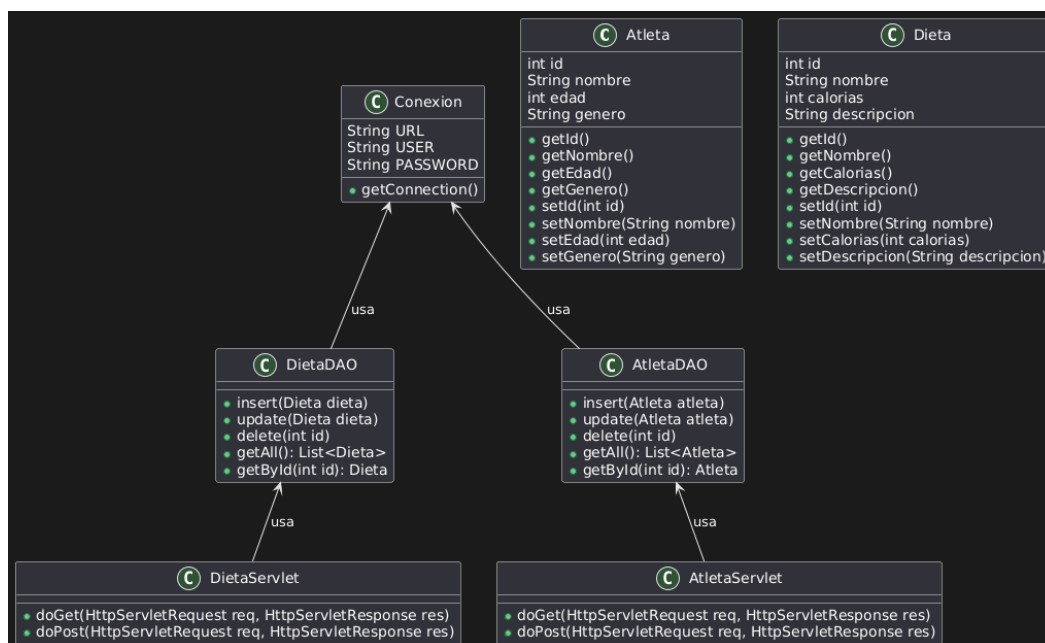
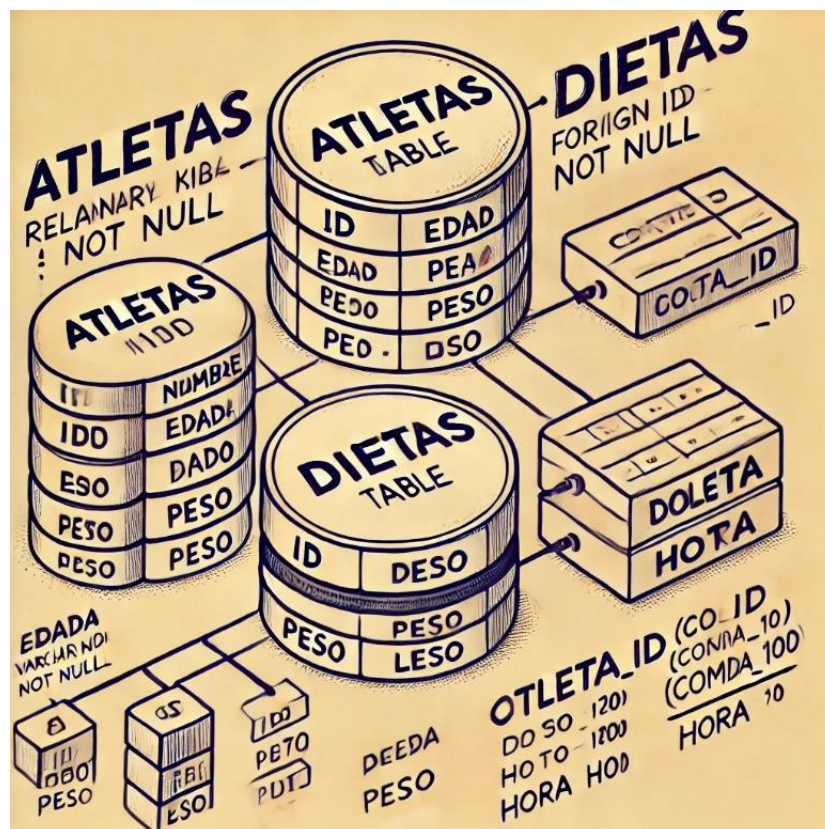
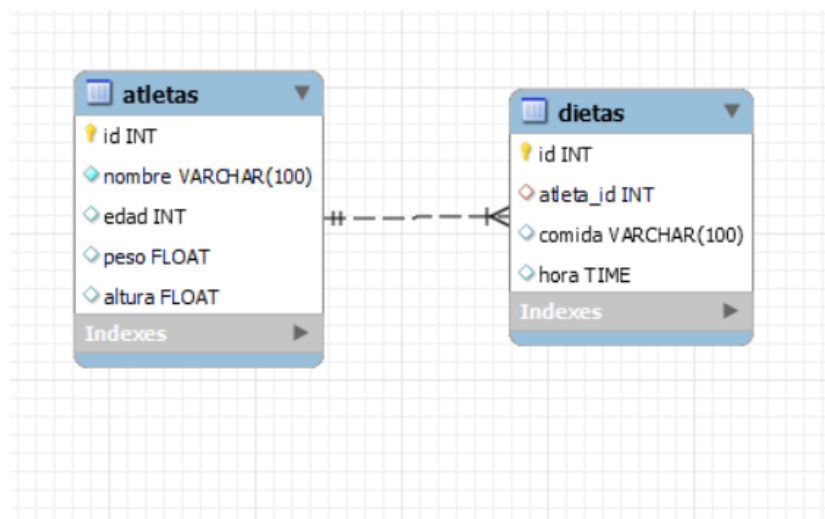


Diagrama modelo relacional:



Diccionario de datos:

Tabla Atleta

Columna	Tipo de Datos	Restricciones	Descripción
Id	INT	PRIMARY KEY, AUTO_INCREMENT	Identificador único de cada atleta
nombre	VARCHAR (100)	NOT NULL	Nombre completo del atleta.
edad	INT		Edad del atleta en años.
peso	FLOAT		Peso atleta en kilogramos.
altura	FLOAT		Altura atleta en metros.

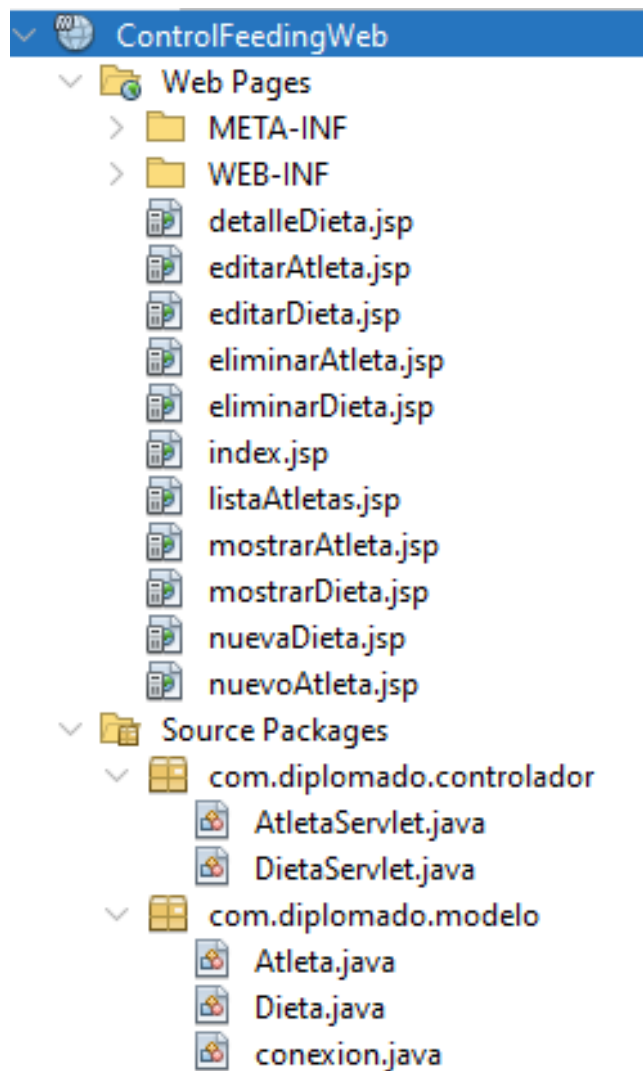
Tabla Comida

Columna	Tipo de Datos	Restricciones	Descripción
Id	INT	PRIMARY KEY, AUTO_INCREMENT	Identificador único de cada registro de dieta.
atleta_id	INT	NOT NULL, FOREIGN KEY REFERENCES atletas(id) ON DELETE CASCADE	Referencia al atleta asociado a este registro de dieta.
comida	VARCHAR (100)	NOT NULL	Nombre o descripción de la comida.

hora	TIME	NOT NULL	Hora programada para consumir la comida.
------	------	----------	--

ESTRUCTURA DEL PROYECTO CREADO EN IDE

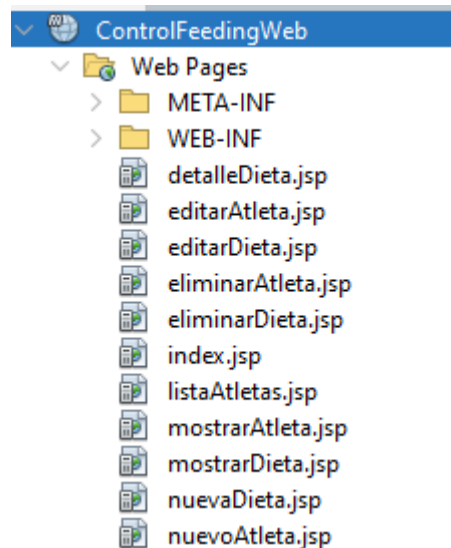
Esquema (imagen) del proyecto creado en NetBeans.



Descripción de cada una de las capas del patrón MVC.

- i. Capa vista: descripción general de cada una de las vistas del sistema: colocar código, imagen JSP

Vistas (View):



- index.jsp: Página principal que muestra la lista de atletas y dietas.

```
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<!DOCTYPE html>
<html>
<head>
    <title>Control Feeding Web</title>
</head>
<body>
    <h1>Bienvenido a Control Feeding Web</h1>
    <p>Seleccione una opción para comenzar:</p>

    <ul>
        <li><a href="listaAtletas.jsp">Gestión de Atletas</a></li>
        <li><a href="detalleDieta.jsp">Gestión de Dietas</a></li>
    </ul>
</body>
</html>
```

listaAtleta

```
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>
    <title>Lista de Atletas</title>
</head>
<body>
    <h1>Lista de Atletas</h1>
    <a href="nuevoAtleta.jsp">Nuevo Atleta</a>
    <table border="1">
        <tr>
            <th>ID</th>
            <th>Nombre</th>
            <th>Edad</th>
            <th>Peso</th>
            <th>Altura</th>
            <th>Acciones</th>
        </tr>
        <c:forEach var="atleta" items="${listaAtletas}">
            <tr>
                <td>${atleta.id}</td>
                <td>${atleta.nombre}</td>
                <td>${atleta.edad}</td>
                <td>${atleta.peso}</td>
                <td>${atleta.altura}</td>
                <td>
                    <a href="DietaServlet?atletaId=${atleta.id}">Ver Dieta |
                    <a href="editarAtleta.jsp?id=${atleta.id}">Editar</a> |
                    <a href="eliminarAtleta.jsp?AtletaServlet?action=delete&id=${atleta.id}">Eliminar</a>
                </td>
            </tr>
        </c:forEach>
    </table>
    <a href="mostrarAtleta.jsp">Ver Atleta</a>
</body>
</html>
```

detalleDieta.

```
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>
<title>Detalle de la Dieta</title>
</head>
<body>
<h1>Detalle de la Dieta para Atleta ID: ${param.atletaId}</h1>
<a href="nuevaDieta.jsp?atletaId=${param.atletaId}">Nueva Comida</a>
<table border="1">
<tr>
<th>ID</th>
<th>Comida</th>
<th>Hora</th>
<th>Acciones</th>
</tr>
<c:forEach var="dieta" items="${listaDietas}">
<tr>
<td>${dieta.id}</td>
<td>${dieta.comida}</td>
<td>${dieta.hora}</td>
<td>
<a href="editarDieta.jsp?id=1${dieta.id}">Editar</a> |
<a href="eliminarDieta.jsp?DietaServlet?action=delete&id=${dieta.id}&atletaId=${param.atletaId}">Eliminar
</td>
</tr>
</c:forEach>
</table>

</td>
</tr>
</c:forEach>
</table>

<a href="mostrarDieta.jsp">Mostrar Dieta</a>
</body>
</html>
```

- nuevoAtleta.jsp: Página para agregar un nuevo atleta.

```

<%@ page contentType="text/html; charset=UTF-8" language="java" %>

<html>
<head>
<title>Nuevo Atleta</title>
</head>
<body>
<h1>Nuevo Atleta</h1>
<form action="AtletaServlet" method="post">
<input type="hidden" name="action" value="create">
<label for="nombre">Nombre:</label>
<input type="text" id="nombre" name="nombre" required><br>

<label for="edad">Edad:</label>
<input type="number" id="edad" name="edad" required><br>

<label for="peso">Peso (kg):</label>
<input type="number" step="0.01" id="peso" name="peso" required><br>

<label for="altura">Altura (cm):</label>
<input type="number" step="0.01" id="altura" name="altura" required><br>

<input type="submit" value="Crear Atleta">
</form>
<a href="listaAtletas.jsp">Volver a la lista de atletas</a>
</body>
</html>

```

- nuevaDieta.jsp: Página para agregar una nueva dieta a un atleta específico.

```

<%@ page contentType="text/html; charset=UTF-8" language="java" %>

<html>
<head>
<title>Nueva Comida</title>
</head>
<body>
<h1>Nueva Comida para Atleta ID: ${param.atletaId}</h1>
<form action="DietaServlet" method="post">
<input type="hidden" name="action" value="create">
<input type="hidden" name="atletaId" value="${param.atletaId}">

<label for="comida">Comida:</label>
<input type="text" id="comida" name="comida" required><br>

<label for="hora">Hora:</label>
<input type="time" id="hora" name="hora" required><br>

<input type="submit" value="Agregar Comida">
</form>
<a href="detalleDieta.jsp?atletaId=${param.atletaId}">Volver al detalle de la dieta</a>
</body>
</html>

```

- editarAtleta.jsp: Página para editar la información de un atleta.

```

<%@ page language="java" contentType="text/html; charset=UTF-8" %>
<html>
<head>
<title>Editar Atleta</title>
</head>
<body>
<h1>Editar Atleta</h1>
<form action="AtletaServlet" method="post">
<input type="hidden" name="action" value="update">
<input type="hidden" name="id" value="${param.id}">
<label for="nombre">Nombre:</label>
<input type="text" name="nombre" id="nombre" value="${atleta.nombre}" required><br>
<label for="edad">Edad:</label>
<input type="number" name="edad" id="edad" value="${atleta.edad}" required><br>
<label for="peso">Peso:</label>
<input type="number" name="peso" id="peso" value="${atleta.peso}" required><br>
<label for="altura">Altura:</label>
<input type="number" name="altura" id="altura" value="${atleta.altura}" required><br>
<input type="submit" value="Actualizar Atleta">
</form>
<a href="listaAtletas.jsp">Cancelar</a>
</body>
</html>

```

- eliminarAtleta.jsp: Página para confirmar la eliminación de un atleta.

```

<%@ page language="java" contentType="text/html; charset=UTF-8" %>
<pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Eliminar Atleta</title>
</head>
<body>
<h2>Eliminar Atleta</h2>
<p>¿Estás seguro que deseas eliminar el atleta: <strong>${request.getParameter("nombre")}</strong>?</p>
<form action="AtletaServlet" method="post">
<input type="hidden" name="accion" value="eliminar">
<input type="hidden" name="id" value="${request.getParameter("id")}">
<input type="submit" value="Eliminar">
<a href="listaAtletas.jsp">Cancelar</a>
</form>
</body>
</html>

```

- mostrarAtleta.jsp: Página para mostrar los detalles de un atleta específico.

```

<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>
    <title>Detalle de Atleta</title>
</head>
<body>
    <h1>Detalle de Atleta</h1>
    <p>ID: ${atleta.id}</p>
    <p>Nombre: ${atleta.nombre}</p>
    <p>Edad: ${atleta.edad}</p>
    <p>Peso: ${atleta.peso}</p>
    <p>Altura: ${atleta.altura}</p>
    <a href="listaAtletas.jsp">Volver a la lista</a>
</body>
</html>

```

- editarDieta.jsp: Página para editar los detalles de una dieta.

```

<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>
    <title>Editar Comida</title>
</head>
<body>
    <h1>Editar Comida para Atleta ID: ${param.atletaId}</h1>
    <form action="DietaServlet" method="post">
        <input type="hidden" name="action" value="update">
        <input type="hidden" name="id" value="${param.id}">
        <input type="hidden" name="atletaId" value="${param.atletaId}">
        <label for="comida">Comida:</label>
        <input type="text" name="comida" id="comida" value="${dieta.comida}" required><br>
        <label for="hora">Hora:</label>
        <input type="time" name="hora" id="hora" value="${dieta.hora}" required><br>
        <input type="submit" value="Actualizar Comida">
    </form>
    <a href="detalleDieta.jsp?atletaId=${param.atletaId}">Cancelar</a>
</body>
</html>

```

- eliminarDieta.jsp: Página para confirmar la eliminación de una dieta.

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Eliminar Dieta</title>
</head>
<body>
    <h2>Eliminar Dieta</h2>
    <p>¿Estás seguro que deseas eliminar esta dieta?</p>

    <form action="DietaServlet" method="post">
        <input type="hidden" name="accion" value="eliminar">
        <input type="hidden" name="id" value="<%= request.getParameter("id") %>">
        <input type="submit" value="Eliminar">
        <a href="detalleDieta.jsp">Cancelar</a>
    </form>
</body>
</html>

```

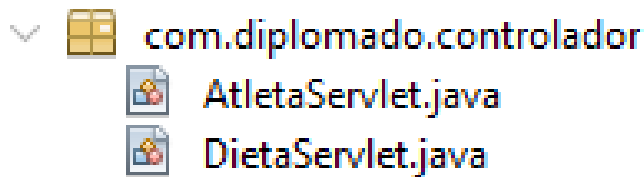
- mostrarDieta.jsp: Página para mostrar los detalles de una dieta específica.

```

<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>
    <title>Detalle de Dieta</title>
</head>
<body>
    <h1>Detalle de Dieta</h1>
    <p>ID: ${dieta.id}</p>
    <p>Comida: ${dieta.comida}</p>
    <p>Hora: ${dieta.hora}</p>
    <a href="detalleDieta.jsp?atletaId=${dieta.atletaId}">Volver a la lista</a>
</body>
</html>

```


- ii. Capa controladora: descripción general del funcionamiento por cada Servlet utilizado, colocar código.



- AtletaServlet: Controlador que maneja las operaciones CRUD para los atletas.

```
package com.diplomado.controlador;
```

```
import com.diplomado.modelo.Atleta;  
import com.diplomado.modelo.conexion;
```

```
import jakarta.servlet.ServletException;  
import jakarta.servlet.annotation.WebServlet;  
import jakarta.servlet.http.HttpServlet;  
import jakarta.servlet.http.HttpServletRequest;  
import jakarta.servlet.http.HttpServletResponse;  
import java.io.IOException;  
import java.sql.Connection;  
import java.sql.PreparedStatement;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.util.ArrayList;  
import java.util.List;
```

```
@WebServlet("/AtletaServlet")  
public class AtletaServlet extends HttpServlet {
```

```
    @Override  
    protected void doPost(HttpServletRequest request, HttpServletResponse  
response) throws ServletException, IOException {  
        // Logica para agregar, editar o eliminar atletas  
        String action = request.getParameter("action");
```

```
        try (Connection conn = conexion.getConnection()) {  
            if (null != action) switch (action) {
```

```

        case "create":{
            String nombre = request.getParameter("nombre");
            int edad = Integer.parseInt(request.getParameter("edad"));
            float peso = Float.parseFloat(request.getParameter("peso"));
            float altura =
Float.parseFloat(request.getParameter("altura"));
            String sql = "INSERT INTO atletas (nombre, edad, peso,
altura) VALUES (?, ?, ?, ?)";
            try (PreparedStatement stmt = conn.prepareStatement(sql)) {
                stmt.setString(1, nombre);
                stmt.setInt(2, edad);
                stmt.setFloat(3, peso);
                stmt.setFloat(4, altura);
                stmt.executeUpdate();
            } break;
        }
// Código para actualizar un atleta
        case "update":
            break;
        case "delete":{
            int id = Integer.parseInt(request.getParameter("id"));
            String sql = "DELETE FROM atletas WHERE id = ?";
            try (PreparedStatement stmt = conn.prepareStatement(sql)) {
                stmt.setInt(1, id);
                stmt.executeUpdate();
            } break;
        }
        default:
            break;
    }
} catch (SQLException e) {
}
//eliminar
if ("delete".equals(action)) {
    int id = Integer.parseInt(request.getParameter("id"));
    String sql = "DELETE FROM atletas WHERE id = ?";
    try (Connection conn = conexion.getConnection();
        PreparedStatement stmt = conn.prepareStatement(sql)) {
        stmt.setInt(1, id);
        stmt.executeUpdate();
    } catch (SQLException e) {

```

```

    }
    response.sendRedirect("listaAtletas.jsp");
}

    response.sendRedirect("listaAtletas.jsp");
}

@Override
protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
    // Mostrar lista de atletas
    List<Atleta> atletas = new ArrayList<>();
    try (Connection conn = conexion.getConnection()) {
        String sql = "SELECT * FROM atletas";
        try (PreparedStatement stmt = conn.prepareStatement(sql)) {
            ResultSet rs = stmt.executeQuery();
            while (rs.next()) {
                Atleta atleta = new Atleta(
                    rs.getInt("id"),
                    rs.getString("nombre"),
                    rs.getInt("edad"),
                    rs.getFloat("peso"),
                    rs.getFloat("altura")
                );
                atletas.add(atleta);
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }

    request.setAttribute("listaAtletas", atletas);
    request.getRequestDispatcher("listaAtletas.jsp").forward(request,
response);
}
}

```

- DietaServlet: Controlador que maneja las operaciones CRUD para las dietas.

```
package com.diplomado.controlador;
```

```
import com.diplomado.modelo.Dieta;
import com.diplomado.modelo.conexion;

import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;
```

```
@WebServlet("/DietaServlet")
public class DietaServlet extends HttpServlet {
```

```
    @Override
```

```
    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
```

```
        // Logica para agregar, editar o eliminar dietas
```

```
        String action = request.getParameter("action");
```

```
        try (Connection conn = conexion.getConnection()) {
```

```
            if (null != action) switch (action) {
```

```

        case "create":{
            int atletaId = Integer.parseInt(request.getParameter("atletaId"));
            String comida = request.getParameter("comida");
            String hora = request.getParameter("hora");
            String sql = "INSERT INTO dietas (atleta_id, comida, hora) VALUES
(?) , ?, ?)";

            try (PreparedStatement stmt = conn.prepareStatement(sql)) {
                stmt.setInt(1, atletaId);
                stmt.setString(2, comida);
                stmt.setString(3, hora);
                stmt.executeUpdate();
            } break;
        }

// Código para actualizar una dieta
        case "update":
            break;
        case "delete":{
            int id = Integer.parseInt(request.getParameter("id"));
            String sql = "DELETE FROM dietas WHERE id = ?";
            try (PreparedStatement stmt = conn.prepareStatement(sql)) {
                stmt.setInt(1, id);
                stmt.executeUpdate();
            } break;
        }
        default:
            break;
    }
} catch (SQLException e) {

```

```
}
```

```
//Eliminar
```

```
if ("delete".equals(action)) {  
    int id = Integer.parseInt(request.getParameter("id"));  
    String sql = "DELETE FROM dietas WHERE id = ?";  
    try (Connection conn = conexion.getConnection();  
        PreparedStatement stmt = conn.prepareStatement(sql)) {  
        stmt.setInt(1, id);  
        stmt.executeUpdate();  
    } catch (SQLException e) {  
    }  
    response.sendRedirect("detalleDieta.jsp?atletaId=" +  
request.getParameter("atletaId"));  
}
```

```
    response.sendRedirect("detalleDieta.jsp?atletaId=" +  
request.getParameter("atletaId"));  
}
```

```
@Override
```

```
protected void doGet(HttpServletRequest request, HttpServletResponse  
response) throws ServletException, IOException {
```

```
    // Mostrar detalle de la dieta para un atleta
```

```
    int atletaId = Integer.parseInt(request.getParameter("atletaId"));  
    List<Dieta> dietas = new ArrayList<>();  
    try (Connection conn = conexion.getConnection()) {  
        String sql = "SELECT * FROM dietas WHERE atleta_id = ?";  
        try (PreparedStatement stmt = conn.prepareStatement(sql)) {
```

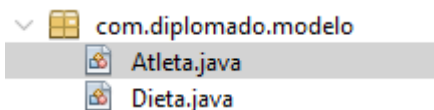
```

        stmt.setInt(1, atletaId);
        ResultSet rs = stmt.executeQuery();
        while (rs.next()) {
            Dieta dieta = new Dieta(
                rs.getInt("id"),
                rs.getInt("atleta_id"),
                rs.getString("comida"),
                rs.getString("hora")
            );
            dietas.add(dieta);
        }
    }
} catch (SQLException e) {
    e.printStackTrace();
}

request.setAttribute("listaDietas", dietas);
request.getRequestDispatcher("detalleDieta.jsp").forward(request, response);
}
}

```

- Capa modelo: descripción general del funcionamiento por cada clase Java para acceso a datos.



```
package com.diplomado.modelo;
```

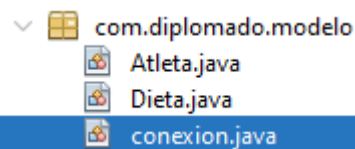
```
public class Atleta {  
    private int id;  
    private String nombre;  
    private int edad;  
    private float peso;  
    private float altura;  
  
    // Constructor, getters y setters  
    public Atleta() {}  
  
    public Atleta(int id, String nombre, int edad, float peso, float altura) {  
        this.id = id;  
        this.nombre = nombre;  
        this.edad = edad;  
        this.peso = peso;  
        this.altura = altura;  
    }  
}
```

- Atleta: Clase que representa a un atleta con atributos como id, nombre, edad, peso y altura.

```
package com.diplomado.modelo;  
  
public class Dieta {  
    private int id;  
    private int atletaId;  
    private String comida;  
    private String hora;  
  
    // Constructor, getters y setters  
    public Dieta() {}  
  
    public Dieta(int id, int atletaId, String comida, String hora) {  
        this.id = id;  
        this.atletaId = atletaId;  
        this.comida = comida;  
        this.hora = hora;  
    }  
}
```

- Dieta: Clase que representa una dieta con atributos como id, atleta_id, comida, y hora.

Conexión con la Base de datos:



```
com.diplomado.modelo  
├── Atleta.java  
├── Dieta.java  
└── conexion.java
```



```

package com.diplomado.modelo;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class conexion {
    private static final String URL = "jdbc:mysql://localhost:3306/control_feeding_web";
    private static final String USER = "root";
    private static final String PASSWORD = "admin";

    public static Connection getConnection() throws SQLException {
        return DriverManager.getConnection(url:URL, user:USER, password:PASSWORD);
    }
}

```

Clase conexion

- La clase "conexión" es la encargada de establecer y proporcionar la conexión con la base de datos.
- Este código es para que las demás partes del proyecto puedan interactuar con la base de datos. Cada vez que se necesite realizar una operación sobre la base de datos (como insertar, actualizar, eliminar o consultar datos), el método getConnection() se invocará para obtener una conexión activa. Esta conexión luego se utiliza para ejecutar las consultas SQL necesarias.

INTEGRACIÓN DEL PROYECTO WEB EN SERVIDOR

Descripción de cada paso y/o configuraciones para el alojamiento del proyecto en servidor.

Para poder manejar el sistema web dentro con el servidor Tomcat, se llevaron a cabo los siguientes pasos:

1. Preparación del Entorno

- **Instalar Apache Tomcat:** Se descargo e instálalo desde el sitio oficial de Apache Tomcat.
- **Configurar las variables de entorno:**
 - Asegúrate de que las variables de entorno JAVA_HOME y CATALINA_HOME estén configuradas correctamente, apuntando a la instalación de Java y Tomcat, respectivamente.

2. Configurar Apache Tomcat en NetBeans

- **Integrar Tomcat con NetBeans:**
 1. Abrimos NetBeans.
 2. Vamos a Servicios (Services) > Servidores (Servers).
 3. Hacemos clic derecho en Servidores y selecciona Añadir Servidor (Add Server).
 4. Seleccionamos Apache Tomcat y luego haz clic en Siguiente (Next).
 5. Especificamos el nombre de instalación del servidor y la ubicación donde se instaló Tomcat (CATALINA_HOME).
 6. Configuramos las credenciales de administrador si es necesario.
 7. Finalizamos el proceso de integración.

3. Desarrollar o importar el Proyecto en NetBeans

- **Crear un nuevo proyecto:** Al desarrollar un nuevo proyecto, seleccionamos Nuevo Proyecto y se eligió Aplicación Web en las categorías de Java EE.

4. Configurar el Proyecto para el Despliegue

- **Configurar el servidor:** Nos aseguramos de que el servidor Tomcat esté seleccionado en la configuración de ejecución del proyecto.
 - Hacemos clic derecho sobre el proyecto en el panel de Proyectos (Projects).

- Seleccionamos Propiedades (Properties) > Ejecutar (Run).
- En la opción de Servidor (Server), selecciona Tomcat.
- **Configurar el contexto del proyecto:**
 - En la misma ventana de Propiedades, podemos configurar el contexto (la URL base) del proyecto, por ejemplo, /miaplicacion.

5. Construir y Desplegar el Proyecto

- **Construir el proyecto:** Hacemos clic derecho sobre el proyecto y selecciona Limpiar y Construir (Clean and Build) para generar el archivo WAR (Web Application Archive).
- **Desplegar en Tomcat:**
 - Desde NetBeans: Una vez que el proyecto esté construido, podemos hacer clic en Ejecutar (Run) para desplegar automáticamente el proyecto en Tomcat.
 - Manualmente: Podemos copiar el archivo WAR generado en la carpeta webapps de la instalación de Tomcat (CATALINA_HOME/webapps).

6. Configurar Apache Tomcat para el Despliegue

- **Ajustar configuraciones en server.xml:**
 - Podemos editar el archivo server.xml de Tomcat para configurar el puerto del servidor, la seguridad, etc. Este archivo se encuentra en CATALINA_HOME/conf.
- **Configuraciones adicionales en context.xml:**
 - Configuramos las propiedades del proyecto o base de datos en el archivo context.xml, ubicado en CATALINA_HOME/conf/context.xml o en la carpeta META-INF del proyecto.

7. Iniciar el Servidor Tomcat

- **Iniciar Tomcat desde NetBeans:** Podemos iniciar y detener el servidor directamente desde NetBeans usando las opciones disponibles en la pestaña de Servicios.
- **Iniciar Tomcat manualmente:** Vamos a la carpeta bin de Tomcat y ejecutamos startup.bat (Windows) o startup.sh (Linux/Mac).

8. Probar la Aplicación

- Una vez desplegado, accedemos a la aplicación desde un navegador usando la URL configurada.

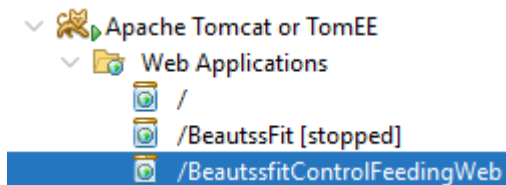
9. Mantenimiento y Gestión

- **Monitorización:** Usamos el Tomcat Manager para gestionar aplicaciones desplegadas, ver logs, y monitorear el rendimiento.
- **Actualizar aplicaciones:** Para actualizar una aplicación, simplemente reemplaza el archivo WAR en webapps y reinicia Tomcat.

Configuración del Servidor de Aplicaciones

Elegir un Servidor de Aplicaciones:

- En este caso, aplicaremos Apache Tomcat.



Se hizo uso de Apache Tomcat, ya que es uno de los servidores web más populares y ampliamente utilizados para aplicaciones Java.

1. Compatibilidad con Servlets y JSP

- Soporte Nativo: Tomcat es un contenedor de servlets que implementa las especificaciones de Java Servlet y JavaServer Pages (JSP). Si la aplicación utiliza estas tecnologías, Tomcat es una opción natural.
- Actualización Constante: Como un proyecto de Apache, Tomcat se mantiene actualizado con las últimas versiones de las especificaciones de Java EE.

2. Ligereza

- Recurso Moderado: Tomcat es más liviano en comparación con servidores de aplicaciones completos como JBoss, WebSphere o WebLogic, lo que lo hace ideal para aplicaciones que no requieren todas las características avanzadas de Java EE.

- **Rápido de Configurar:** Su instalación y configuración iniciales son relativamente sencillas, permitiendo un despliegue más rápido.

3. Popularidad y Comunidad

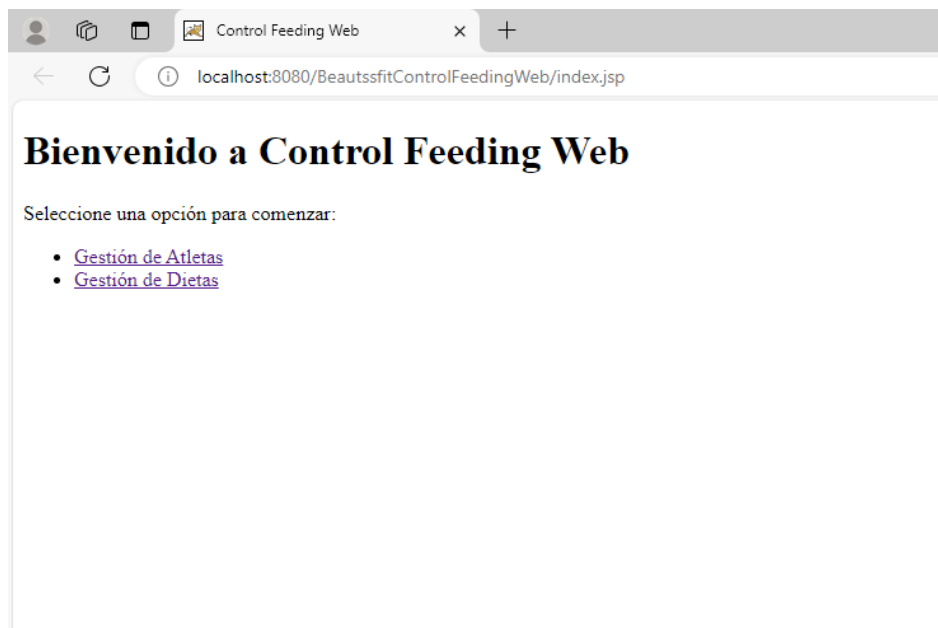
- **Amplia Adopción:** Tomcat es utilizado por muchas empresas y desarrolladores alrededor del mundo, lo que significa que hay una gran cantidad de recursos, tutoriales y soporte disponible.
- **Comunidad Activa:** La comunidad activa alrededor de Tomcat garantiza actualizaciones regulares y soluciones rápidas a problemas comunes.

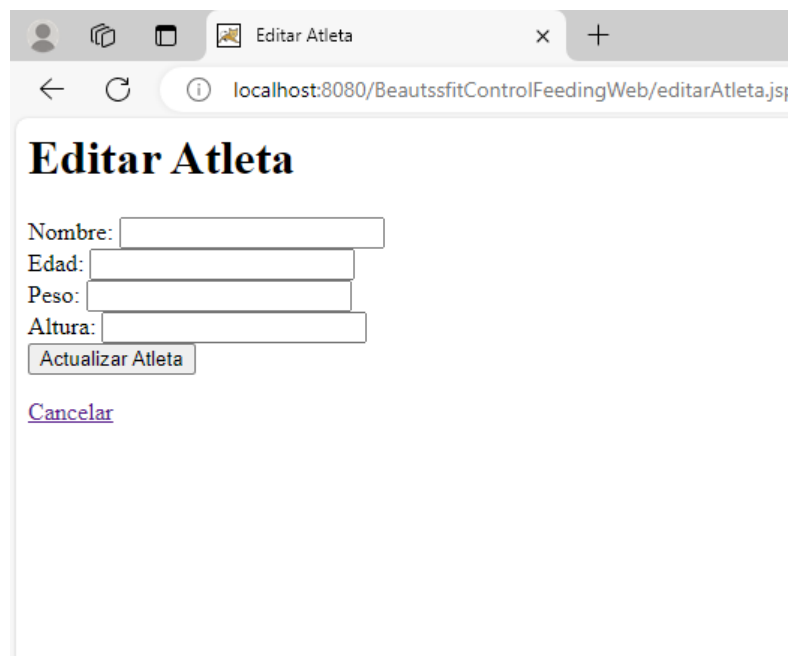
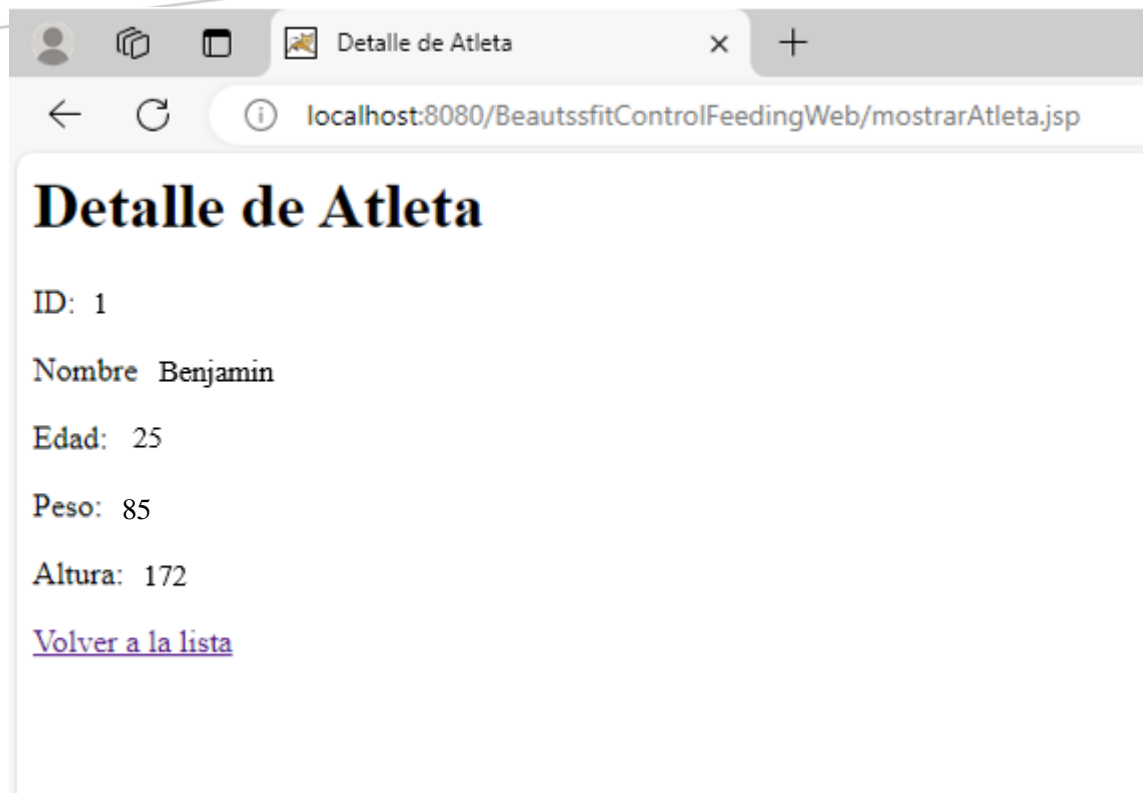
4. Open Source

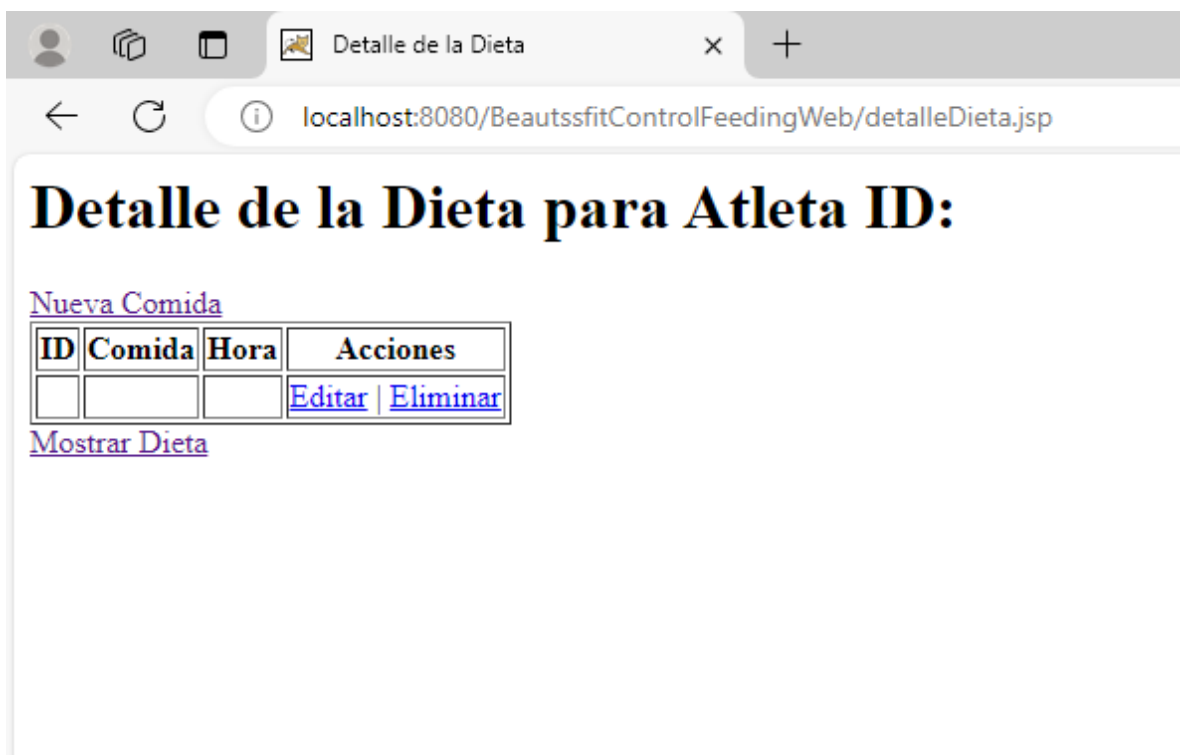
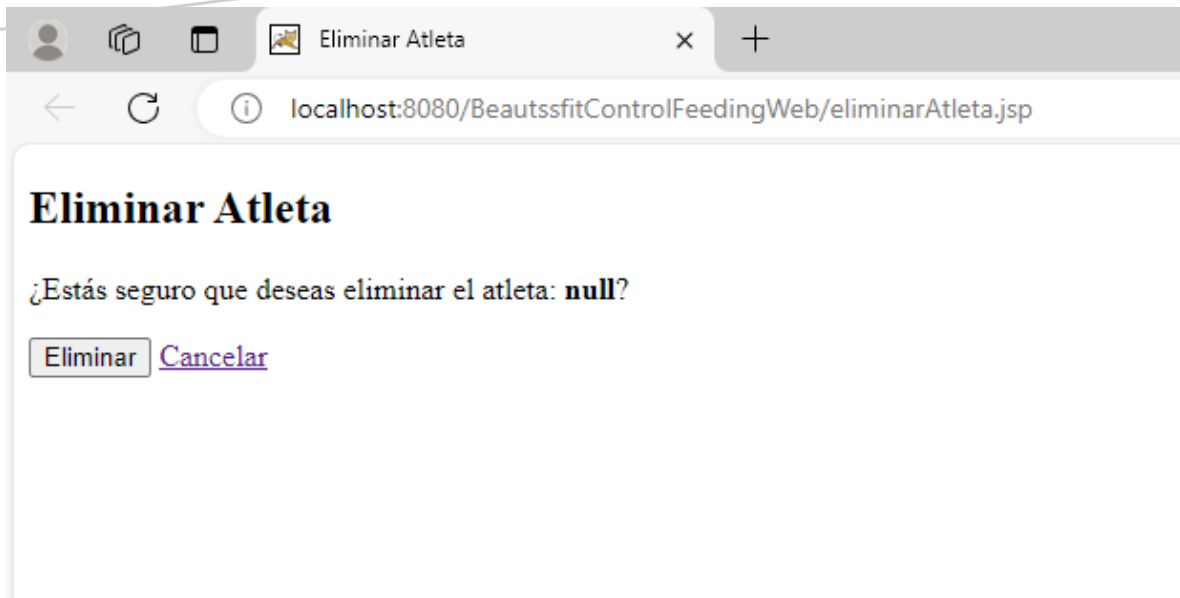
- **Sin Costo:** Tomcat es de código abierto, lo que significa que puedes utilizarlo sin costos de licencia. Esto es atractivo para proyectos pequeños, startups, o empresas que buscan reducir costos.
- **Personalizable:** Como es de código abierto, puedes modificar Tomcat para ajustarlo a tus necesidades específicas.

Se concluye que al usar Tomcat es una opción excelente para aplicaciones Java que requieren un servidor de aplicaciones ligero, eficiente, y compatible con las especificaciones de servlets y JSP, especialmente si por la búsqueda de un balance entre simplicidad y funcionalidad sin incurrir en altos costos.

Imágenes ilustrativas del correcto funcionamiento del sistema web en servidor.







Nueva Comida

localhost:8080/BeautssfitControlFeedingWeb/nuevaDieta.jsp?atletaId=

Nueva Comida para Atleta ID:

Comida:

Hora: 

[Volver al detalle de la dieta](#)

Detalle de Dieta

localhost:8080/BeautssfitControlFeedingWeb/mostrarDieta.jsp

Detalle de Dieta

ID: 1

Comida: Proteina

Hora: 9:08 AM

[Volver a la lista](#)

Editar Comida para Atleta ID:

Comida:

Hora:

[Cancelar](#)

Eliminar Dieta

¿Estás seguro que deseas eliminar esta dieta?

[Cancelar](#)

FUENTES CONSULTADAS

Álvarez, L. Á. (2009). *DISEÑO Y DESARROLLO DE UNA*. Madrid.