

Aplicación web dinámica con patrón DAO y manipulación de datos

Aplicación web dinámica con patrón DAO y manipulación de datos	1
¿Qué aprenderás?	2
Introducción	2
Estructurar vista bootstrap a un proyecto JSP	3
Creando la tabla de usuarios	6
Añadiendo funcionalidad a login.jsp para login	6
Servlet procesaLogin	8
Construcción de filtros de departamentos	12
Creación de clases DAO	16
Interface BusquedaEmpleadoDao	16



¡Comencemos!

¿Qué aprenderás?

- Estructurar vista bootstrap en un proyecto jsp.
- Añadir funcionalidad a login.jsp para login.
- Generar un Servlet procesaLogin.
- Construir filtros de las entidades.
- Crear clases DAO.

Introducción

Previamente, ya se ha trabajado en una arquitectura modelo vista controlador, utilizando un servlet como controller, los jsp como vista y el modelo con las clases de negocio, los cuales se apoyan de un patrón DAO para el acceso a datos. Si bien todo funciona, es un ejemplo básico, ya que solo hacemos una consulta simple y la desplegamos por pantalla.

Ahora es cuando empiezan casos algo más complejos, ya que en la realidad no solo se utiliza una tabla, sino que muchas y relacionadas mediante técnicas de modelado de datos según las necesidades del negocio.

Se implementará una nueva vista para la interfaz de usuario, más acorde a una aplicación web. Nuestro trabajo será adaptarla a nuestras necesidades. Descargamos el template de bootstrap llamado [SB Admin 2](#) y comenzamos a trabajar.

Estructurar vista bootstrap a un proyecto JSP

Esta plantilla tiene el siguiente aspecto (la imagen es referencial):

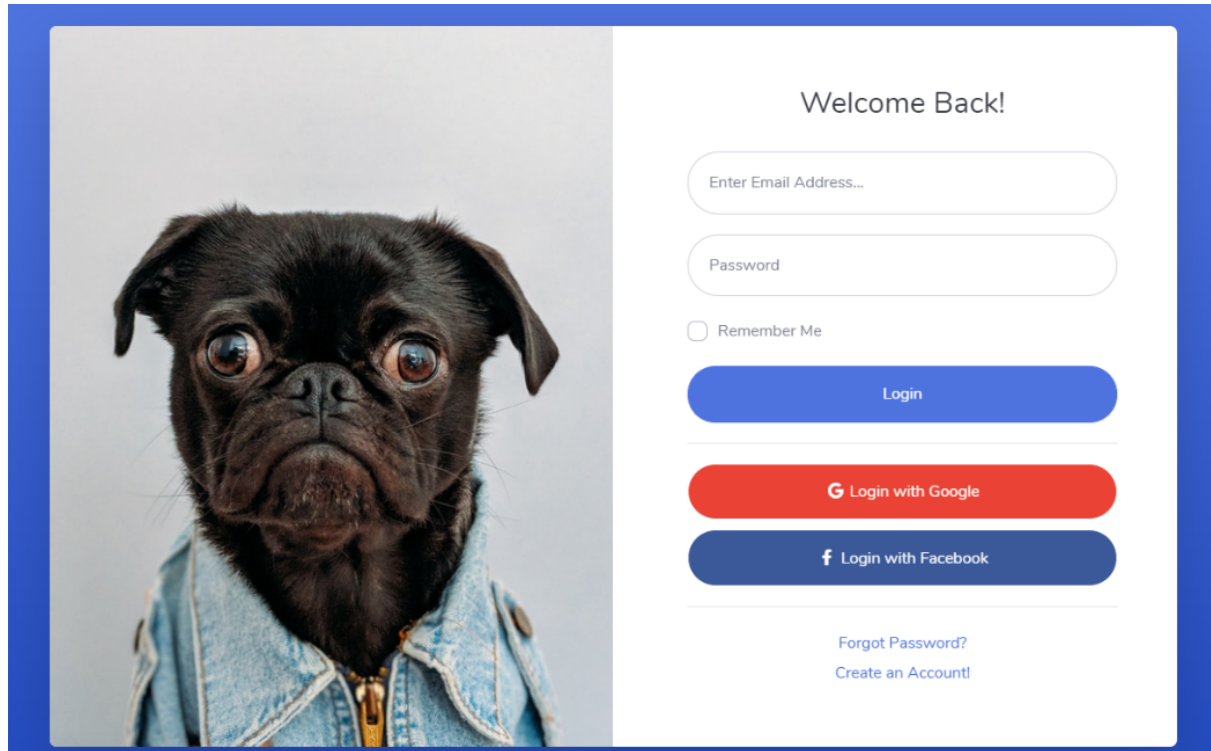


Imagen 1. Plantilla Bootstrap.
Fuente: Desafío Latam.

Para utilizar la plantilla seguir los siguientes pasos:

1. En el proyecto *ControlaDeptosEmpresa*, abrir la carpeta *WebContent* y en ella copiar todo el contenido de la carpeta de la plantilla que se acaba de instalar. Se espera que la carpeta quede con esta estructura:

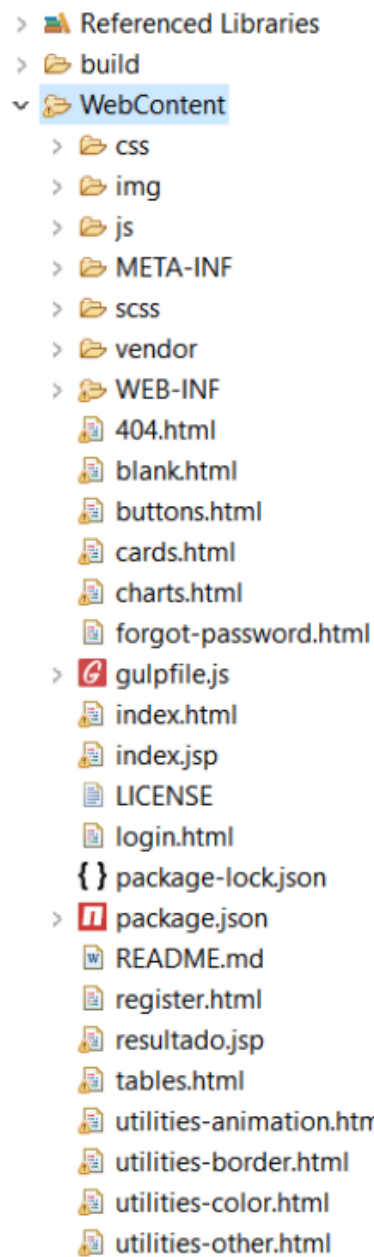


Imagen 2. Carpeta WebContent.
Fuente: Desafío Latam.

2. Ahora, es momento de indicar al proyecto por qué página debe comenzar. Para eso ubicar el archivo `web.xml` dentro de la carpeta `WebContent-WEB-INF`.

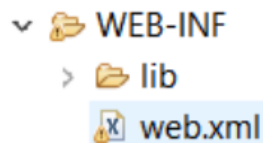


Imagen 3. Archivo `web.xml`.

Fuente: Desafío Latam.

3. Este archivo nos permite configurar una welcome page que se abrirá al momento de iniciar la aplicación. Para generar este comportamiento agregar las siguientes líneas:

```
1
2 <web-app>
3   <welcome-file-list>
4     <welcome-file>login.html</welcome-file>
5   </welcome-file-list>
6   <resource-ref>
7     <description>Pool conexiones</description>
8     <res-ref-name>jdbc/ConexionOracle</res-ref-name>
9     <res-type>javax.sql.DataSource</res-type>
10    <res-auth>Container</res-auth>
11  </resource-ref>
12 </web-app>
```

Imagen 4. Configurando la página de bienvenida.

Fuente: Desafío Latam.

4. En la línea 4 indicamos que la página de bienvenida será la `login.html`. Ahora, al iniciar nuevamente la aplicación el server apache sabrá qué página abrir y desplegará el login.

5. Estas páginas están en formato html y para efectos del curso debemos convertirlas en páginas JSP. Para lograr hacerlo se debe añadir las configuraciones de una página jsp (en la imagen) sobre la etiqueta **<!DOCTYPE html>** del archivo.

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
```

6. Para luego simplemente cambiar la extensión de login.html a login.jsp.
7. Ahora en el archivo Web.xml hacer lo mismo. Con esas configuraciones la página html se convierte en una página JSP.

Creando la tabla de usuarios

Se procede a crear una nueva tabla en la base de datos que almacene los valores de los usuarios administradores. La actual relación de datos corresponde a 1 a N. Significa que un usuario puede tener varias cuentas de usuario y una cuenta de usuario puede pertenecer a una sola persona.

```
CREATE TABLE USUARIOS_ADMIN (ID_USUARIO NUMBER PRIMARY KEY, CORREO
VARCHAR(100), PASS VARCHAR(100), PRIMER_NOMBRE VARCHAR(100),
SEGUNDO_NOMBRE VARCHAR(100));
```

Con esta tabla es posible generar un inicio de sesión.

Añadiendo funcionalidad a login.jsp para login

La plantilla tiene listas las vistas para iniciar sesión, para hacer un nuevo registro, etc. Nosotros deseamos que la aplicación permita iniciar sesión mediante un correo y una contraseña que el usuario ingresará en la página login.jsp. El procesamiento de la información estará a cargo de un servlet de nombre procesaLogin que a su vez se comunicará con las clases del patrón DAO las cuales tendremos que crear. El flujo simplificado se grafica en el diagrama:

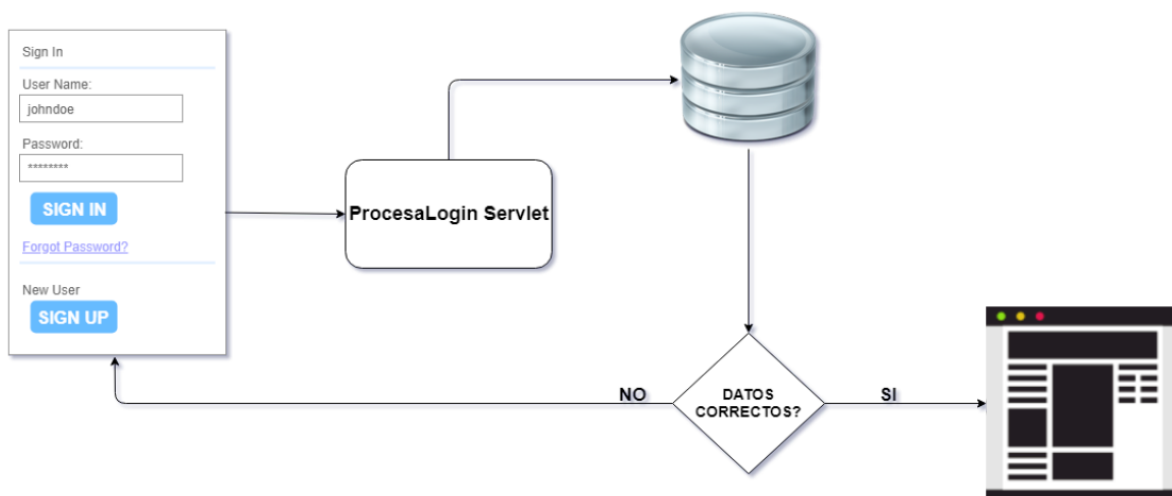


Imagen 5. Flujo para iniciar sesión.

Fuente: Desafío Latam.

Para iniciar el login, primero se trabaja sobre la vista. El archivo login.jsp cuenta con un formulario HTML normal. Para que trabaje junto al servlet y pueda enviarle datos, es necesario configurar las etiquetas form con:

- Etiqueta action="procesaLogin"
- Etiqueta method="get"

Etiqueta Action para enviar los valores al servlet con el mismo nombre de identificación.
Etiqueta method para indicar al servidor el tipo de envío de datos.

El archivo *login.jsp* se muestra a continuación, dando énfasis solo a las líneas que modificamos para que trabaje junto al servlet.

```

<form class="user" action="procesaLogin" method="GET">
  <div class="form-group">
    <input type="email" name="email" class="form-control form-control-user" id="exampleInputEmail" aria-describedby="emailHelp" placeholder="Email" />
  </div>
  <div class="form-group">
    <input type="password" name="password" class="form-control form-control-user" id="exampleInputPassword" placeholder="Password" />
  </div>
  <div class="form-group">
    <div class="custom-control custom-checkbox small">
      <input type="checkbox" class="custom-control-input" id="customCheck">
      <label class="custom-control-label" for="customCheck">Remember Me</label>
    </div>
  </div>
  <input type="submit" value="Login" class="btn btn-primary btn-user btn-block">
</form>

```

Imagen 6. Agregando login al JSP.

Fuente: Desafío Latam.

Importante mencionar que el botón de envío debe de ser de tipo submit. Estas modificaciones permiten que desde la página JSP se envíen los datos de correo y password al servlet.

Servlet procesaLogin

Luego de modificar la capa de vista, se modifica la capa controladora, representada en este caso por el servlet ProcesaLogin. Este servlet recibe el request y captura los valores de correo y password enviados desde la vista, para luego enviar los datos a las capas interiores, específicamente a las clases DAO.

```
30  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
31      boolean usuarioExiste = false;
32      String correo = request.getParameter("email");
33      String password = request.getParameter("password");
34      LoginDaoImp loginDao = new LoginDaoImp();
35      usuarioExiste = loginDao.usuarioRegistrado(correo,password);
36      request.setAttribute("usuarioExiste", usuarioExiste);
37      if(usuarioExiste) {
38          HttpSession sesionUsuario = request.getSession(true);
39          sesionUsuario.setAttribute("correo", correo);
40          request.setAttribute("correo", correo);
41          request.getRequestDispatcher("index.jsp").forward(request, response);
42          return;
43      }
44      //request.setAttribute("deptoDao", listaDeptos);
45      request.getRequestDispatcher("login.jsp").forward(request, response);
46  }
```

Imagen 7. Modificando el controlador ProcesaLogin.

Fuente: Desafío Latam.

Mediante el método `getParameter`, se obtienen los valores enviados desde el formulario para luego enviarlos al método `usuario Registrado` correspondiente a la clase `LoginDao`. Recordemos que dentro del patrón DAO, existe una interfaz que define los métodos que las clases deben implementar.

En este caso la interface `loginDao` implementa solamente el metodo `usuarioRegistrado` que recibe como parámetros el correo y las password ingresadas por el usuario.

```
1  package com.desafiolatam.dao;
2
3  public interface LoginDao {
4      public boolean usuarioRegistrado(String correo, String password);
5  }
```

Imagen 8. LoginDao.

Fuente: Desafío Latam.

Cada interfaz DAO va acompañada por una clase concreta que implementa los métodos definidos en la interfaz. Para el login se crea una clase que implementa la interfaz de nombre LoginDaoImpl.java.

```
18 @Override
19 public boolean usuarioRegistrado(String correo, String password) {
20     boolean usuarioExiste = false;
21     String sql = "SELECT * FROM USUARIOS_ADMIN WHERE CORREO = '" + correo + "'" + " AND PASS = '" + password + "'";
22     try {
23         pstmt = conn.prepareStatement(sql);
24         rs = pstmt.executeQuery();
25         if(rs.next()) {
26             usuarioExiste = true;
27         }
28     } catch (SQLException e) {
29         // TODO Auto-generated catch block
30         e.printStackTrace();
31     }
32     return usuarioExiste;
33 }
34 }
```

Imagen 9. Implementando LoginDao.

Fuente: Desafío Latam.

Esta clase es quién se comunica con la base de datos así que se encarga de generar un select a la tabla recién creada USUARIOS ADMIN en busca de coincidencias. Tales coincidencias se buscan mediante una sentencia select, que busca la igualdad entre correo y password en la cláusula where.

El método retorna un booleano, el cual es TRUE si se encontraron coincidencias y FALSE en caso de no encontrar al usuario registrado en la base de datos. Tal variable booleana es comprobada desde el servlet, el cual dependiendo del resultado redirige al index del sistema o a la pantalla de inicio de sesión para que la gente la ingrese nuevamente.

```
/**
 * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
 */
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    boolean usuarioExiste = false;
    String correo = request.getParameter("email");
    String password = request.getParameter("password");
    LoginDaoImpl loginDao = new LoginDaoImpl();
    usuarioExiste = loginDao.usuarioRegistrado(correo,password);
    request.setAttribute("usuarioExiste", usuarioExiste);
    if(usuarioExiste) {
        HttpSession sesionUsuario = request.getSession(true);
        sesionUsuario.setAttribute("correo", correo);
        request.setAttribute("correo", correo);
        request.getRequestDispatcher("index.jsp").forward(request, response);
        return;
    }
    //request.setAttribute("deptoDao", listaDeptos);
    request.getRequestDispatcher("login.jsp").forward(request, response);
}
```

Imagen 10. Modificando doGet.

Fuente: Desafío Latam.

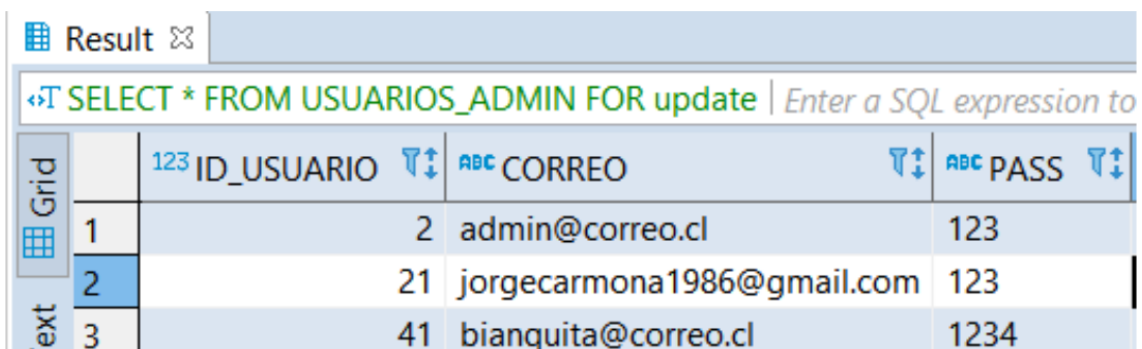
El servlet también crea una variable de sesión la cual registra el correo para luego validar la presencia de un usuario registrado en el sistema. A continuación analizaremos el jsp de nombre index.jsp.

```
35<body id="page-top">
36  <%
37      String correo = "no existe usuario";
38      HttpSession misesion = request.getSession();
39      if (mision != null) {
40          correo = (String) misesion.getAttribute("correo");
41      } else {
42          request.getRequestDispatcher("login.jsp").forward(request, response);
43      }
44  %>
```

Imagen 11. index.jsp.
Fuente: Desafío Latam.

La línea 40 obtiene el valor '**correo**' que se asignó en el servlet para guardarla en la variable del mismo nombre. Esto es básicamente para mostrar el correo que inició sesión en la aplicación, y así demostrar que el inicio de sesión se ejecutó correctamente.

Actualmente en la tabla USUARIOS_ADMIN existen algunos registros de usuarios, de los cuales usaremos a admin@correo.cl y su pass 123.



	123 ID_USUARIO	ABC CORREO	ABC PASS
1	2	admin@correo.cl	123
2	21	jorgecarmona1986@gmail.com	123
3	41	bianquita@correo.cl	1234

Imagen 12. Tabla USUARIOS_ADMIN.
Fuente: Desafío Latam.

En la aplicación iniciamos sesión:

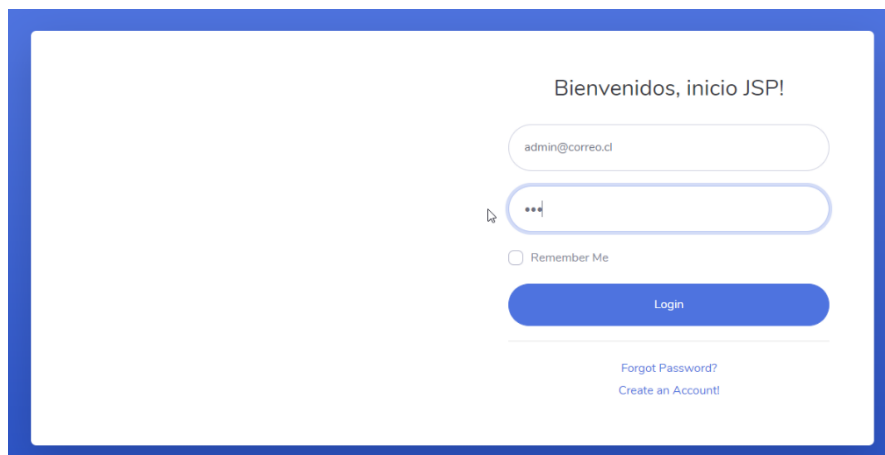


Imagen 13. Iniciando sesión.
Fuente: Desafío Latam.

Al presionar el botón Login, el servlet luego de comprobar los valores en la base de datos redirige al jsp index y si nos fijamos en la parte superior se puede ver el correo que se ingresó en el formulario.

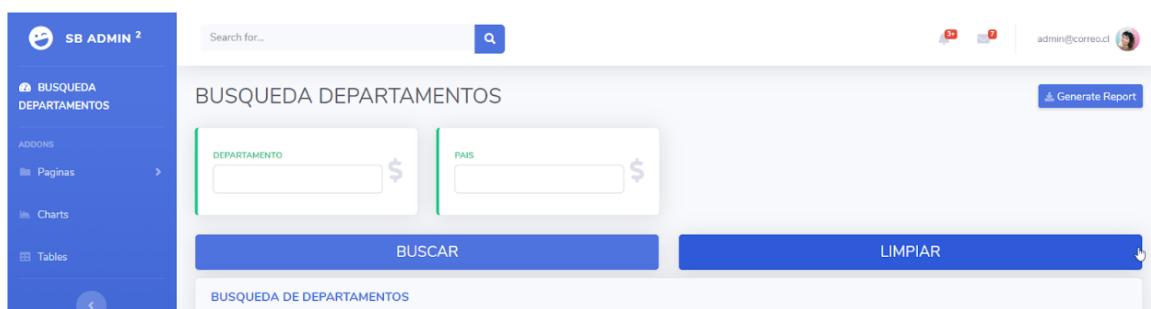


Imagen 14. Inicio correcto.
Fuente: Desafío Latam.

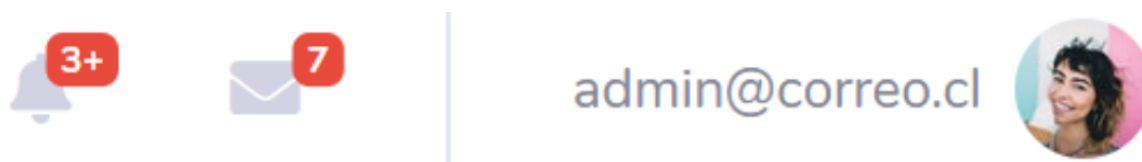


Imagen 15. Usuario conectado.
Fuente: Desafío Latam.

Cabe aclarar que las imágenes son referenciales y no nos preocupamos de ellas, ya que excede los alcances de la unidad, pero se pueden modificar sin problemas.

Para desplegar un valor rescatado de base de datos en la vista html, simplemente debemos capturar la variable y luego con anotaciones jsp vistas en capítulos anteriores la utilizamos.

En la imagen de a continuación se grafica la porción de código que hace tal tarea con la variable `<%=correo%>`:

```
292 <!-- Nav Item - User Information -->
293 <li class="nav-item dropdown no-arrow"><a
294     class="nav-link dropdown-toggle" href="#" id="userDropdown"
295     role="button" data-toggle="dropdown" aria-haspopup="true"
296     aria-expanded="false">
297     <span>class="mr-2 d-none d-lg-inline text-gray-600 small"><%=correo%></span>
298     
```

Imagen 16. Variable correo.

Fuente: Desafío Latam.

Con estas acciones implementamos un login utilizando la relación 1 a N mediante jdbc y patrón DAO.

Construcción de filtros de departamentos

En un sistema web enfocado a la empresa, se habla mucho de los mantenedores que tienen la gran misión de dar mantenimiento a los datos alojados en las tablas de base de datos. Uno de los aspectos más relevantes es la vista que despliega los datos a base de filtros. Los filtros son parámetros de las tablas en la cual restringimos los resultados a lo que el usuario desee ver.

Para construir un filtro, primero debemos definir cuáles son los parámetros de la tabla que se utilizaran para las búsquedas.

SELECT * FROM EMPLEADO | Enter a SQL expression to filter results (use Ctrl+Space)

	123	NUMEMPLEADO	ABC	NOMBRE	123	NUMDEPTO
1		101		Matias Zarate		11
2		16		Matias Zarate		15
3		17		Felipe Perez		15
4		18		Bob Marley		15
5		19		Pedro Polanco		15
6		20		Eduardo Azul		15
7		21		Oscar Fernandez		14

Imagen 17. Definir los parámetros que se utilizarán para la búsqueda.

Fuente: Desafío Latam.

Se planea implementar filtros básicos utilizando una query sencilla. Por ejemplo, si queremos ver a todos los empleados que pertenezcan al departamento número 14 se debe pensar en la sentencia:

```
select * from empleados where numdepto = 14;
```

Ahora, si además se desea obtener a todos los empleados del departamento 14 y que se llame Pedro Polanco la query aumenta algo de complejidad:

```
select * from empleados where numdepto = 14 and nombre = 'Pedro Polanco' ;
```

En estos casos da igual si se obtienen registros o no, lo importante es que la aplicación debe estar lista para poder hacer consultas con distintos criterios y relaciones. A continuación se implementará el buscador de empleados con jsp-servlet bajo el patrón DAO y MVC.

Recordar que estamos trabajando con una plantilla bootstrap que se está modificando según nuestras necesidades. Solo nos enfocaremos en el código necesario para llevar a cabo las tareas encomendadas.

Primero, vamos a modificar el archivo index.jsp y en él vamos a agregar tres input text y dos botones (buscar y limpiar) todo esto dentro de un formulario que enviará los datos mediante el método get.

Imagen 18. Modificando el index.jsp.

Fuente: Desafío Latam.

A primera vista, hay bastante código que es parte de la plantilla bootstrap, pero solo tomar en cuenta la línea 351 que corresponde a un input text con id="nombre" y name="nombre". En este bloque de código creamos un campo de texto que enviará el nombre al servlet.

```

343 <!-- Earnings (Monthly) Card Example -->
344 <div class="col-xl-3 col-md-6 mb-4">
345   <div class="card border-left-success shadow h-100 py-2">
346     <div class="card-body">
347       <div class="row no-gutters align-items-center">
348         <div class="col mr-2">
349           <div
350             class="text-xs font-weight-bold text-success text-uppercase mb-1">Nombre Empleado</div>
351           <input type="text" class="form-control" id="nombre" name="nombre">
352         </div>
353         <div class="col-auto">
354           <i class="fas fa-dollar-sign fa-2x text-gray-300"></i>
355         </div>
356       </div>
357     </div>
358   </div>
359 </div>

```

Imagen 19. Creando campo de texto.

Fuente: Desafío Latam.

Entonces, generamos dos elementos input text adicionales duplicando los bloques de código y generando los inputs correspondientes, quedando con NOMBRE EMPLEADO, N° EMPLEADO y N° DEPARTAMENTO.

Imagen 20. Creando campos nombre de empleado, número de empleado y número de departamento.

Fuente: Desafío Latam.

No olvidar que enviaremos estos datos al servlet mediante un request http de tipo get. Para esto, tanto los input text como los botones deben estar encerrados entre etiquetas form.

325

```
<form action="procesaBusquedaEmpleado" method="get">
```

Imagen 21. ProcesaBúsquedaEmpleado.

Fuente: Desafío Latam.

Y el botón debe ser de tipo submit para que los datos viajen al servidor.

```
<button type="submit" class="btn btn-primary btn-lg btn-block">BUSCAR</button>
```

Imagen 22. Creando botón Buscar.

Fuente: Desafío Latam.

Con el jsp modificado, vamos a generar el servlet. Desde aquí ya creamos código desde cero.

En eclipse dentro del *package com.desafiolatam.servlet* un *servlet* de nombre *ProcesaBusquedaEmpleado.java*.

```
ProcesaBusquedaEmpleado.java
1 package com.desafiolatam.servlet;
2
3 import java.io.IOException;
4
5 /**
6  * Servlet implementation class ProcesaLogin
7  */
8 @WebServlet("/procesaBusquedaEmpleado")
9 public class ProcesaBusquedaEmpleado extends HttpServlet {
10     private static final long serialVersionUID = 1L;
11
12     /**
13      * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
14      */
15     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
16         List<Empleado> listaEmpleados = new ArrayList<Empleado>();
17         String nombre = request.getParameter("nombre");
18         String numEmpleado = request.getParameter("numEmpleado");
19         String numDepartamento = request.getParameter("numDepto");
20
21         if(numEmpleado.isEmpty()) {
22             numEmpleado = "0";
23         }
24
25         if(numDepartamento.isEmpty()) {
26             numDepartamento = "0";
27         }
28
29         BusquedaEmpleadoDaoImp busqueda = new BusquedaEmpleadoDaoImp();
30         listaEmpleados = busqueda.busquedaDepto(nombre, Integer.parseInt(numEmpleado), Integer.parseInt(numDepartamento));
31         request.setAttribute("empleadoDao", listaEmpleados);
32         request.getRequestDispatcher("busquedaEmpleados.jsp").forward(request, response);
33     }
34 }
```

Imagen 23. Servlet ProcesaBusquedaEmpleado.java.

Fuente: Desafío Latam.

En la línea 47 se genera una instancia de la clase *BusquedaEmpleadoDaoImp.java* la cual devuelve una lista de empleados mediante el método *busquedaEmpleado.java*.

Es importante destacar que estamos en el controlador de la aplicación y tal como se vio en él previamente, enviaremos los datos a la abstracción de datos. Notar que este proyecto es

una versión aumentada del proyecto base generado anteriormente. Recordemos el diagrama del patrón DAO.

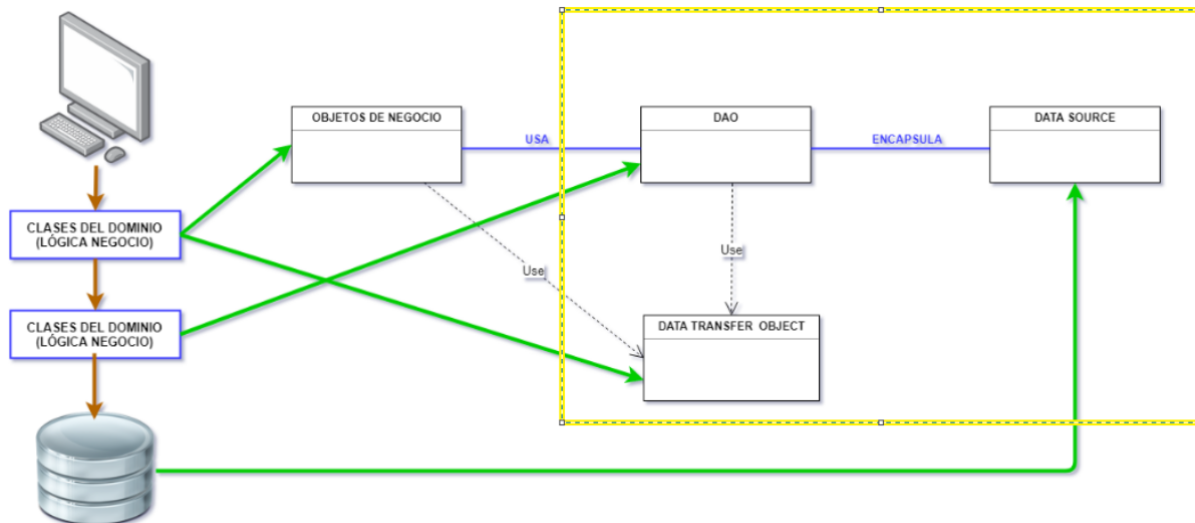


Imagen 24. Diagrama patrón DAO.
Fuente: Desafío Latam.

Creación de clases DAO

Dentro del package `com.desafiolatam.dao` crear una interface y una clase de nombre:

- `BusquedaEmpleadoDao.java`
- `BusquedaEmpleadoDaoImp.java`

Interface `BusquedaEmpleadoDao`

```
1 package com.desafiolatam.dao;
2
3 import java.util.List;
4
5
6
7
8 public interface BusquedaEmpleadoDao {
9     public List<Empleado> busquedaEmpleado(String nombre, int numEmpleado, int numDepartamento);
10 }
```

Imagen 25. `BusquedaEmpleadoDao`.
Fuente: Desafío Latam.

Se crean los métodos sin implementación pero con sus parámetros de entrada. En este caso se crea el método `busquedaEmpleado` que recibe el nombre, `numEmpleado`,

numDepartamento. A continuación se genera la implementación mediante la clase *BusquedaEmpleadoDaoImp.java*.

```
11 import java.sql.Connection;
12 public class BusquedaEmpleadoDaoImp extends AdministradorConexion implements BusquedaEmpleadoDao {
13     public BusquedaEmpleadoDaoImp() {
14         Connection conn = generaPoolConexion();
15     }
16     @Override
17     public List<Empleado> busquedaEmpleado(String nombre, int numEmpleado, int numDepartamento) {
18         String query = "SELECT * FROM EMPLEADO WHERE ";
19         List<Empleado> empleados = new ArrayList<Empleado>();
20         if((nombre.isEmpty()) && (numEmpleado == 0) && (numDepartamento == 0)) {
21             query = "SELECT * FROM EMPLEADO";
22         } else {
23             if(!nombre.isEmpty() && (numEmpleado > 0) && (numDepartamento > 0)) {
24                 query.concat("NOMBRE = '"+nombre+"' AND NUMEMPLEADO = "+numEmpleado+ " AND " + " NUMDEPTO = " + numDepartamento);
25             } else {
26                 if(!nombre.isEmpty()) {
27                     query += "NOMBRE = '"+nombre+"'";
28                 }
29                 if(numEmpleado > 0){
30                     query += "NUMEMPLEADO = "+numEmpleado;
31                 }
32                 if(numDepartamento > 0){
33                     query += "NUMDEPTO = "+numDepartamento;
34                 }
35             }
36         }
37         try {
38             pstmt = conn.prepareStatement(query);
39             rs = pstmt.executeQuery();
40             while(rs.next()) {
41                 Empleado empleado = new Empleado(rs.getInt("NUMEMPLEADO"),rs.getString("NOMBRE"),rs.getInt("NUMDEPTO"));
42                 empleados.add(empleado);
43             }
44         } catch (SQLException e) {
45             // TODO Auto-generated catch block
46             e.printStackTrace();
47         }
48         return empleados;
49     }
50 }
51 }
```

Imagen 26. Implementación de la clase *BusquedaEmpleadoDaoImp.java*.

Fuente: Desafío Latam.

En el método verificamos la existencia de los valores de entrada y mediante sentencias de decisión se arma la *query*. Al final se devuelve una lista de Empleados al *servlet*, el cual dependiendo de la respuesta redirige al *jsp busquedaEmpleados.jsp*.

En la vista generamos una tabla normal, pero con código java entre las etiquetas de la línea 463 y 468.

```

444@ <div class="table-responsive">
445@ <table class="table table-bordered" id="dataTable"
446@ width="100%" cellspacing="0">
447@ <thead>
448@ <tr>
449@ <th>Nombre Empleado</th>
450@ <th>Numero Empleado</th>
451@ <th>Numero Departamento</th>
452@ </tr>
453@ </thead>
454@ <tfoot>
455@ <tr>
456@ <th>Nombre Empleado</th>
457@ <th>Numero Empleado</th>
458@ <th>Numero Departamento</th>
459@ </tr>
460@ </tfoot>
461@ <tbody>
462@ <%
463@ if (request.getAttribute("empleadoDao") != null) {
464@ List<Empleado> empleados;
465@ empleados = (List) request.getAttribute("empleadoDao");
466@
467@ for (Empleado empleado : empleados) {
468@ %>
469@ <tr>
470@ <td><%=empleado.getNombreEmpleado()%></td>
471@ <td><%=empleado.getNumEmpleado()%></td>
472@ <td><%=empleado.getNumDepto()%></td>
473@ </tr>
474@ <%
475@ }
476@ }
477@ %>
478@ </tbody>
479@ </table>

```

Imagen 27. Generación de tabla normal.
Fuente: Desafío Latam.

Como resultado, el sistema permite hacer búsquedas a la tabla de departamentos mediante filtros.

Nombre Empleado	Numero Empleado	Numero Departamento
Adamiro Rojas	62	6
Alex Citrico	58	7
Alfonso Mastil	60	7
Arturo Prat	70	5
Aston Gonzalez	79	3
Astorga Daniel	32	12
Bianca Vicencio	43	10
Bob Marley	18	15

Imagen 28. Búsqueda de tablas con filtros.
Fuente: Desafío Latam.

Como se pudo observar, añadir una nueva funcionalidad en una estructura bien definida resulta sencillo. Solamente se tuvo que implementar:

- En la vista generar los controles que capturan y muestran la información (los input text, botones y formulario para el envío y una tabla para el despliegue.)
- En el controlador, un servlet que trabaja como puerta de entrada a la aplicación, una interfaz dao y una implementación, la cual se encarga de la lógica que se comunica con la base de datos.

Recordamos esta receta, ya que como se puede ver el sistema está estructurado de tal forma que cada responsabilidad de las clases está bien marcada e implementar nuevas funcionalidades siempre tendrá esta forma mecánica de construcción, aunque puede variar la lógica del DAO o las validaciones.

En la implementación de *empleadoDao* se puede ver una query, la cual trabaja directamente con el modelo de datos y sus relaciones. Esta sección de código es en donde se implementan las sentencias SQL que capturan los datos, y si bien la lógica de tratamiento puede variar, la sentencia sql es quien hace el trabajo y es fija.