Arquitectura y Diseño de sistemas Web y C/S

Práctica Final

Convocatoria Ordinaria

(Evaluación continua)

Curso 2020/2021

Marina Benito Silvestre

Álvaro Blanco Álvarez

Cristina Elena Dascalu

Alberto González Martínez

Aarón Iglesias Chorro

Contenido

ANÁLISIS DEL PROBLEMA

Este proyecto trata sobre una empresa, que tiene una plantilla personal que pone a disposición de distintos clientes, en el cual se necesita desarrollar un sistema informático para la gestión de los costes del personal.

La aplicación está dividida en dos grupos, el personal de recursos humanos y el personal de los clientes.

Cuando se habla del personal de recursos humanos debe tener el control de las empresas, proyectos, trabajadores y calendarios, así como atender todas las peticiones de los trabajadores. En cualquier momento se podrá solicitar a través de la aplicación un informe sobre el cumplimiento de las jornadas de trabajo de la empresa, proyecto o empleado.

Cuando hablamos de la aplicación, nos referimos a llevar un control diario sobre el trabajo, los empleados harán un marcaje de su entrada y salida, dirán las jornadas realizadas en el día anterior en base al proyecto, indican el numero de horas de la jornada que dedicaran a cada proyecto asignado, también solicitaran vacaciones, horas, días libres en el calendario.

Estos cambios lo tienen que aprobar el personal de recursos humanos, una vez conseguida esta aprobación el empleado recibirá un mensaje con la respuesta.

PROBLEMAS ENCONTRADOS Y SOLUCIONES APORTADAS

La mayoría del tiempo lo dedicamos a investigar bastante sobre JSP y servlets ya que no entendíamos bien su funcionamiento. Nos costó unir la parte de Java con la de HTML

En la parte de base de datos algún problema que hemos tenido fue la de hacer la conexión.

Otro de los problemas que hemos tenido con la base de datos ha sido a la hora de hacer el calendario con la clave primaria, no podíamos generar solicitudes porque la clave se repetía, entonces lo que pensamos al final fue poner una doble.

En la parte de fichar empleado el problema que tuvimos es que no funcionaba cuando le metíamos un dato tipo date. Tuvimos que separar la fecha y la hora con el método substring y así poder coger los datos de fecha y hora por separado.

También tuvimos problemas a la hora de los botones de eliminar y editar ya que los dos necesitaban el ID pero en el form solo se podía poner un action

Otro problema encontrado fue que no sabíamos cómo avisar al usuario de que se la solicitud que ha hecho se ha aprobado o no, lo que pensamos fue crear un sistema de alertas que cuando se cargue la página de inicio de usuario coge todas las solicitudes tramitadas y no leídas y genera una alerta.

La base de datos se ha realizado en MySQL. Para la realización del diagrama hemos usado Enterprise Architect, porque creemos que es una solución muy eficiente y fácil de utilizar ya que se ha usado en cursos anteriores.

IMPLEMENTACIÓN

Para la elaboración de este software se ha utilizado el patrón Modelo-Vista-Controlador. En la clase Modelo hemos incluido las funcionalidades del sistema y clases con sus respectivos métodos para la conexión con la base de datos. La vista, por su parte, incluye las páginas HTML con sus formularios con los cuales el cliente puede interactuar. Por último, en el controlador hemos incluido Servlets para capturar las peticiones del cliente y comunicarlas al modelo para ofrecer las vistas adecuadas.

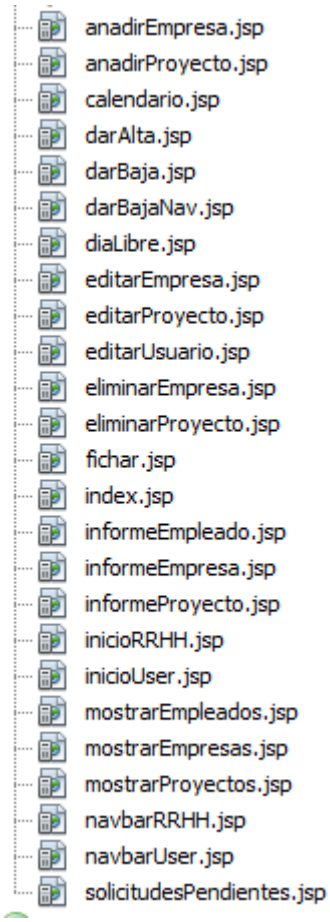

Figura 1. Vista

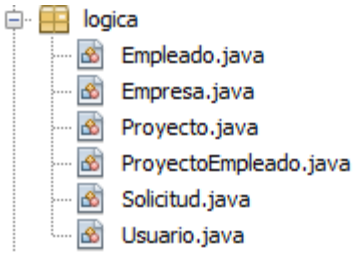

Figura 2. Modelo

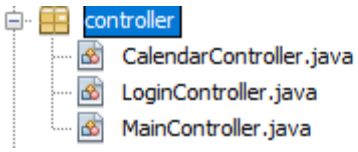

Figura 3. Controlador

La parte del Modelo, como hemos dicho anteriormente, se conecta a la base de datos mediante unas clases encargadas de dicha conexión. Además, el software implementa funciones propias de la parte de recursos humanos y funciones propias de la parte de empleados normales. A continuación, vamos a ver más en detalle estas funcionalidades.

Implementación Recursos Humanos

Mostrar Empleados

Se ha implementado un método para mostrar todos los empleados dados de alta en la base de datos.

```
public List mostrarEmpleados() {
    ArrayList<Usuario> lista_empleados = new ArrayList<>();
    Log.logBd.info("CONSULTA MostrarEmpleados");
    try {
        conexion = ConexionBd.getConexion();
        Log.logBd.info("Realizada conexion - mostrarEmpleados()");
        Statement s = conexion.createStatement();
        ResultSet resultado = s.executeQuery("select * from empleadoempresa");
        Log.logBd.info("Realizada consulta - mostrarEmpleados()");

        while (resultado.next()) {
            Usuario usuario = new Usuario();

            usuario.setIdUsuario(resultado.getInt("IdEmpleadoEmpresa"));
            usuario.setNombre(resultado.getString("Nombre"));
            usuario.setApellidos(resultado.getString("Apellidos"));
            usuario.setEmail(resultado.getString("Correo"));
            usuario.setTelefono(resultado.getInt("Telefono"));
            usuario.setContrasenna(resultado.getString("Contrasenia"));
            List<ProyectoEmpleado> lista = getListaProyectos(resultado.getString("Correo"));
            usuario.setProyectosList(lista);

            if (usuario.getProyectosList().size() > 0) {
                usuario.setEmpresa(usuario.getProyectosList().get(0).getProyecto().getEmpresa());
            }

            lista_empleados.add(usuario);
        }
    } catch (SQLException error) {
        Log.logBd.error("ERROR SQL en mostrarEmpleados(): " + error);
```

Figura 4. mostrarEmpleados

Por otro lado, tenemos una vista con un botón Mostrar Empleados que incluye una referencia a otra vista mostrarEmpleados.jsp.

```
    <a href="./mostrarEmpleados.jsp"><span class="fa fa-users" style="font-size:180px;color: #bd043b;text-align:center
</div>
```

Figura 5. Botón mostrar Empleados

Cuando el usuario pulsa el botón, el controlador captura la acción y realiza la operación correspondiente que, en este caso, implica mostrar al usuario la vista correspondiente.

```
if(accion.equalsIgnoreCase("mostrarEmpleados")){
    siguientePagina = MOSTRAR_EMPLEADOS;
}
```

Figura 6. Capturar acción y devolver resultado.

Posteriormente, se abre una nueva vista con la lista de empleados. En esta vista representada mediante un jsp llamamos al método mostrarEmpleados.

```
<%
    ConsultaBd emp = new ConsultaBd();
    List<Usuario> lista_empleados = emp.mostrarEmpleados(); //lista con los empleados de la base de datos
    Iterator<Usuario> iterador = lista_empleados.iterator();
    Usuario u = null;
    //para cada empleado se genera una fila
    while (iterador.hasNext()) { //recorre la lista de empleados
        u = iterador.next();
%>
<!--cuerpo de la tabla-->
<tbody>
    <tr> <!-- inicio de una fila de la tabla-->
        <td>
            <input type="radio" name="empleado" value="<%=u.getIdUsuario()%>" required> <!--seleccionar usuario a
        </td>
        <td><%= u.getIdUsuario()%></td>
        <td><%= u.getNombre()%></td>
        <td><%= u.getApellidos()%></td>
        <%  String nombreEmpresa="";
            Empresa empresa;
            empresa =u.getEmpresa();
            if(empresa == null){
                nombreEmpresa = "Empleado no registrado correctamente";
            }else{
                nombreEmpresa = empresa.getNombre();
            }
        %>
        <td><%= nombreEmpresa %></td>
        <td><%= u.getEmail()%></td>
        <td><%= u.getTelefono()%></td>
```

Figura 7. Vista de los empleados

En esta misma vista, el usuario tiene opciones de añadir un nuevo empleado, eliminarlo o editarlo.

Añadir

Para añadir empleados se ha implementado el método darAlta(Usuario). Este método se basa en hacer un insert en la base de datos de un objeto de tipo usuario con unos atributos definidos previamente.

```
public boolean darAlta(Usuario usuario) {
    Log.logBd.info("CONSULTA DarAlta");
    boolean hecho = false;
    try {
        conexion = ConexionBd.getConexion();
        Log.logBd.info("Realizada conexion - darAlta()");

        Statement s = conexion.createStatement();
        ResultSet resultado = s.executeQuery("select Correo from empleadoempresa");
        List<String> listaCorreos = new ArrayList<>();
        while (resultado.next()) {
            listaCorreos.add(resultado.getString("Correo"));
        }

        //Si no existe el usuario lo damos de alta
        if (!listaCorreos.contains(usuario.getEmail())) {
            Statement s1 = conexion.createStatement();
            s1.executeUpdate("INSERT INTO EmpleadoEmpresa(IdEmpleadoEmpresa, Nombre, Apellidos, Telefono, Correo, Contrasenia) VA
                    + "','" + usuario.getApellidos() + "','" + usuario.getTelefono() + "','" + usuario.getEmail() + "','" + usua
            Statement s2 = conexion.createStatement();
            //Insertamos el proyecto en el que se ha dado de alta al usuario
            Proyecto proyecto = usuario.getProyectosList().get(0).getProyecto();
            s2.executeUpdate("INSERT INTO proyecto_empleado(Horas, proyecto_id_proyecto, empleado_correo) VALUES (" + 0 + "," + p
            Log.logBd.info("Usuario correo(" + usuario.getEmail() + ") dado de alta");
            hecho = true;
        } else {
            Log.logBd.error("Usuario correo(" + usuario.getEmail() + ") ya esta dado de alta");
        }
```

Figura 8. Dar Alta

Al presionar el botón añadir desde la vista anterior, el usuario obtiene un formulario donde ha de ingresar los datos del usuario que desea dar de alta. Al presionar el botón Enviar, el

controlador recibe la acción y llama al método darAlta pasándole un objeto de tipo usuario con los atributos del formulario que nuestro usuario ha enviado.

```
if(accion.equalsIgnoreCase("altaEmpleado")){
    Usuario usuario = new Usuario();
    usuario = consulta.generarId(usuario);
    String nombre = request.getParameter("nombre");
    String apellidos = request.getParameter("apellidos");
    String telefono = request.getParameter("telefono");
    String correo = request.getParameter("correo");
    String pass = request.getParameter("password");
    int idProyecto = Integer.parseInt(request.getParameter("idProyecto"));
    usuario.setNombre(nombre);
    usuario.setApellidos(apellidos);
    usuario.setTelefono(Integer.parseInt(telefono));
    usuario.setEmail(correo);
    usuario.setContrasenna(pass);
    //Añadimos el nuevo proyecto del empleado
    Empresa empresa = consulta.getEmpresaProyecto(idProyecto);
    Proyecto proyecto = new Proyecto();
    proyecto.setIdProyecto(idProyecto);
    proyecto.setEmpresa(empresa);
    ProyectoEmpleado proyectoEmpleado = new ProyectoEmpleado();
    proyectoEmpleado.setProyecto(proyecto);
    proyectoEmpleado.setHoras(0);
    List<ProyectoEmpleado> listaProyectos = new ArrayList<>();
    listaProyectos.add(proyectoEmpleado);
    usuario.setProyectosList(listaProyectos);
    usuario.setEmpresa(empresa);
    boolean agregar=consulta.darAlta(usuario);

    if(agregar){
        sesion.setAttribute("mensaje", "Usuario "+ usuario.getEmail() +" dado de alta correctamente");
```

Figura 9. Controlador captura la acción altaEmpleado

Eliminar

Para eliminar un empleado tenemos el método darBaja(correo) que realiza una consulta en la base de datos para buscar el usuario que tiene asignado dicho correo. Si lo encuentra, se realiza una consulta para eliminar el usuario de la base de datos.

```
public boolean darBaja(String correo) {
    boolean hecho = false;
    Log.logBd.info("CONSULTA DarBaja");
    try {
        conexion = ConexionBd.getConexion();
        Log.logBd.info("Realizada conexion - darBaja()");
        Statement s = conexion.createStatement();
        ResultSet resultado = s.executeQuery("select * from EmpleadoEmpresa where Correo='" + correo + "';");

        //Si existe le damos de baja y borramos sus entradas en la tabla proyecto_empleado
        if (resultado.next()) {
            s.executeUpdate("delete from proyecto_empleado where empleado_correo='" + correo + "';");
            int codigo = s.executeUpdate("delete from EmpleadoEmpresa where Correo='" + correo + "';");
            if (codigo > 0) {
                Log.logBd.info("Usuario correo(" + correo + ") dado de baja");
                hecho = true;
            } else {
                Log.logBd.info("No se han podido borrar los proyecto del usuario correo(" + correo + ")");
            }
        } else {
            Log.logBd.error("Usuario no existe en la BD");
        }
    } catch (SQLException error) {
        Log.logBd.error("ERROR SQL en darBaja(): " + error);
        Log.logBd.error("                       SQL State - " + error.getSQLState());
        Log.logBd.error("                       ErrorCode - " + error.getErrorCode());
    }
```

Figura 10. darBaja

El usuario selecciona un empleado a eliminar desde la vista y, esta acción, se pasa al controlador. El controlador obtiene de la vista el correo del empleado a eliminar y realiza una llamada al método darBaja pasándole como parámetro el correo obtenido.

```
else if(accion.equalsIgnoreCase("bajaEmpleado")){
    String correo = request.getParameter("correo");
    boolean baja=consulta.darBaja(correo);

    if(baja){
        sesion.setAttribute("mensaje", "Usuario "+ correo +" dado de baja correctamente");
    }
    else{
        sesion.setAttribute("mensaje", "ERROR: No se pudo dar de baja al usuario "+ correo);
    }

    siguientePagina=MOSTRAR_EMPLEADOS;
```

Figura 11. Controlador captura acción bajaEmpleado

Editar

Para editar un empleado se ha utilizado el método modificarUsuario que recibe como parámetros los nuevos valores del usuario a modificar. Se realiza un UPDATE en la base de datos de los valores que se desean modificar del usuario identificado por su correo.

```
public boolean modificarUsuario(int idUsuario, String nombre, String apellidos, int telefono, String correo, String contrasenna)
    Log.logBd.info("CONSULTA ModificarUsuario");
    boolean hecho = false;
    try {
        conexion = ConexionBd.getConexion();
        Log.logBd.info("Realizada conexion - modificarUsuario()");
        Statement s = conexion.createStatement();
        int codigo = s.executeUpdate("update empleadoempresa set Nombre ='" + nombre + "', Apellidos='" + apellidos + "', Telefo
                + ", Correo='" + correo + "', Contrasenia='" + contrasenna + "' where Correo='" + correo + "';");

        //Si la consulta se ha realizado correctamente
        if (codigo > 0) {
            hecho = true;
            Log.logBd.info("Realizada consulta - modificarUsuario()");
        } else {
            Log.logBd.info("Consulta no ha alterado la tabla o consulta errónea - modificarUsuario() - cod." + codigo);
        }
    } catch (SQLException error) {
        Log.logBd.error("ERROR SQL en modificarUsuario(): " + error);
        Log.logBd.error("                                  SQL State - " + error.getSQLState());
        Log.logBd.error("                                  ErrorCode - " + error.getErrorCode());
    }

    return hecho;
}
```

Figura 12. modificarUsuario

El usuario selecciona un empleado desde la vista mostrarEmpleados y pulsa el botón Editar. Este botón genera una nueva vista con un formulario para introducir los nuevos datos del usuario. El controlador captura la acción junto con los nuevos valores introducidos en el formulario y llama al método modificarUsuario pasándole dichos valores.

```
else if(accion.equalsIgnoreCase("editarEmpleado")){
    int idUsuario = Integer.parseInt(request.getParameter("idUsuario"));
    String nombre = request.getParameter("nombre");
    String apellidos = request.getParameter("apellidos");
    int telefono = Integer.parseInt(request.getParameter("telefono"));
    String correo = request.getParameter("correo");
    String contrasenna = request.getParameter("password");

    boolean exito = consulta.modificarUsuario(idUsuario, nombre, apellidos, telefono, correo, contrasenna);

    if(exito){
        siguientePagina = MOSTRAR_EMPLEADOS;
        sesion.setAttribute("mensaje", "Usuario "+ correo +" modificado con exito.");
    }else{
        siguientePagina = EDITAR_EMPLEADO;
        sesion.setAttribute("mensaje", "ERROR: No se ha podido modificar al usuario "+ correo);
    }
}
```

Figura 13. Controlador captura acción editarEmpleado

Mostrar Proyectos

Se ha implementado un método para mostrar todos los proyectos dados de alta en la base de datos.

```
public List mostrarProyecto() {
    ArrayList<Proyecto> lista_proyectos = new ArrayList<>();
    Log.logBd.info("CONSULTA MostrarProyecto");
    try {
        conexion = ConexionBd.getConexion();
        Log.logBd.info("Realizada conexion - mostrarProyecto()");
        Statement s = conexion.createStatement();
        ResultSet resultado = s.executeQuery("select * from proyecto");
        Log.logBd.info("Realizada consulta - mostrarProyecto()");

        while (resultado.next()) {
            Proyecto proyecto = new Proyecto();
            Empresa empresa = new Empresa();

            proyecto.setIdProyecto(resultado.getInt("IdProyecto"));
            empresa.setIdEmpresa(resultado.getInt("Empresa_IdEmpresa"));
            proyecto.setEmpresa(empresa);

            lista_proyectos.add(proyecto);
        }
    } catch (SQLException error) {
        Log.logBd.error("ERROR SQL en mostrarProyecto(): " + error);
        Log.logBd.error("                                 SQL State - " + error.getSQLState());
        Log.logBd.error("                                 ErrorCode - " + error.getErrorCode());
    }

    Log.logBd.info("Consulta realizada con éxito - mostrarProyecto()");
    return lista_proyectos;
}
```

Figura 14. Método mostrar proyectos

Por otro lado, tenemos una vista con un botón Mostrar Proyectos que incluye una referencia a otra vista mostrarProyectos.jsp.

```
<div class="col-md-4 d-flex justify-content-center">
    <a href="./mostrarProyectos.jsp"><span class="fa fa-paperclip" style="font-size:180px;color: #bd043b;text-align:ce
</div>
```

Posteriormente, se abre una nueva vista con la lista de proyectos. En esta vista representada mediante un jsp llamamos al método mostrarProyectos.

```
    </thead>
    <%
        ConsultaBd proyecto=new ConsultaBd();
        List<Proyecto> lista_proyectos=proyecto.mostrarProyecto(); //lee de la BD todos los proyectos existentes
        Iterator<Proyecto> iterador=lista_proyectos.iterator();
        Proyecto p=null;
        while(iterador.hasNext()){ //recorre la lista de proyectos
            p=iterador.next();
    %>
    <tbody>
        <tr> <!-- inicio de una fila de la tabla-->
            <td>
                <input type="radio" name="proyecto" value="<%= p.getIdProyecto() %>" required> <!-- en value poner i
            </td>
            <td><%= p.getIdProyecto() %></td>
            <td><%= p.getEmpresa().getIdEmpresa() %></td>
        </tr>
        <% } %>
    </tbody>
</table>
<div class="col-md-12 text-right">
    <button type="submit" class="btn btn-danger text-right" style="height:40px" onclick="">
        Editar
    </button>
    <button type="submit" formaction="MainController?action=elegirProyecto&boton=eliminar" class="btn btn-danger tex
        Eliminar
    </button>
    <button type="submit" formaction="MainController?action=elegirProyecto&boton=anadir" class="btn btn-danger text-
        Añadir
    </button>
```

Figura 15. Iterar proyecto en la vista

En esta misma vista, el usuario tiene opciones de añadir un nuevo proyecto, eliminarlo o editarlo.

Añadir

Para añadir proyectos se ha implementado el método anadirProyecto. Este método se basa en hacer un insert en la base de datos de un nuevo proyecto.

```
public boolean anadirProyecto(int idProyecto, int idEmpresa) {
    boolean hecho = true;
    Log.logBd.info("CONSULTA AnadirProyecto");
    try {
        conexion = ConexionBd.getConexion();
        Log.logBd.info("Realizada conexion - anadirProyecto()");
        Statement s = conexion.createStatement();

        //Hacemos una lista con los id de las empresas para comprobar que se elige una empresa que esta en la BD
        Statement s1 = conexion.createStatement();
        ResultSet resultado1 = s1.executeQuery("select Empresa_IdEmpresa from proyecto");
        List<Integer> listaEmpresas = new ArrayList<>();
        while (resultado1.next()) {
            listaEmpresas.add(resultado1.getInt("Empresa_IdEmpresa"));
        }

        Statement s2 = conexion.createStatement();
        ResultSet resultado2 = s2.executeQuery("select IdProyecto from proyecto");
        List<Integer> listaProyectos = new ArrayList<>();
        while (resultado2.next()) {
            listaProyectos.add(resultado2.getInt("IdProyecto"));
        }

        if (listaEmpresas.contains(idEmpresa) && !listaProyectos.contains(idProyecto)) {
            int codigo = s.executeUpdate("INSERT INTO proyecto(IdProyecto, Empresa_IdEmpresa) VALUES (" + idProyecto + "," +
            hecho = false;
            if (codigo > 0) {
                Log.logBd.info("Realizada consulta - anadirProyecto()");
```

Figura 16. Método añadir proyecto

Al presionar el botón añadir desde la vista anterior, el usuario obtiene un formulario donde ha de ingresar los datos del proyecto que desea añadir. Al presionar el botón Enviar, el controlador recibe la acción y llama al método anadirProyecto pasándole los atributos del formulario que nuestro usuario ha enviado.

```
else if(accion.equalsIgnoreCase("anadirProyecto")){
    int idProyecto = Integer.parseInt(request.getParameter("idProyecto"));
    int idEmpresa = Integer.parseInt(request.getParameter("idEmpresa"));

    boolean exito = consulta.anadirProyecto(idProyecto, idEmpresa);

    if(exito){
        sesion.setAttribute("mensaje", "Proyecto "+ idProyecto +" añadido con éxito.");
    }else{
        sesion.setAttribute("mensaje", "ERROR: No se ha podido añadir el proyecto "+ idProyecto);
    }

    siguientePagina = MOSTRAR_PROYECTOS;
}
```

Figura 17. Controlador captura acción de añadir proyecto

Eliminar

Para eliminar un proyecto tenemos el método borrarProyecto(id) que realiza una consulta en la base de datos para eliminar los registros de las tablas que dependen del proyecto identificado por el parámetro id. Una vez realizado este paso, se realiza otra consulta para borrar el proyecto de la base de datos.

```
public boolean borrarProyecto(int idProyecto) {
    Log.logBd.info("CONSULTA BorrarProyecto");
    boolean hecho = false;
    try {
        conexion = ConexionBd.getConexion();
        Log.logBd.info("Realizada conexion - borrarProyecto()");
        Statement s = conexion.createStatement();
        Log.logBd.info("Borramos las entrada en la tabla proyecto_empleado");
        s.executeUpdate("delete from proyecto_empleado where proyecto_id_proyecto=" + idProyecto + ";");

        //Se han borrado los proyectos de la tabla proyecto_empleado y borramos el proyecto de la tabla empleado
        int codigo = s.executeUpdate("delete from proyecto where IdProyecto=" + idProyecto + ";");

        if (codigo > 0) {
            hecho = true;
            Log.logBd.info("Proyecto con id(" + idProyecto + ") borrado");
        } else {
            Log.logBd.error("No se ha podido o no se ha encontrado el proyecto id(" + idProyecto + ") - borrarProyecto() en proye
        }

    } catch (SQLException error) {
        Log.logBd.error("ERROR SQL en borrarProyecto(): " + error);
        Log.logBd.error("                               SQL State - " + error.getSQLState());
        Log.logBd.error("                               ErrorCode - " + error.getErrorCode());
    }

    return hecho;
}
```

Figura 18. Método borrar proyecto

El usuario selecciona un proyecto a eliminar desde la vista y, esta acción, se pasa al controlador. El controlador obtiene de la vista el id del proyecto a eliminar y realiza una llamada al método borrarProyecto pasándole como parámetro los datos obtenidos.

```
else if(accion.equalsIgnoreCase("eliminarProyecto")){
    int idProyecto = Integer.parseInt(request.getParameter("idProyecto"));

    boolean borrado = consulta.borrarProyecto(idProyecto);

    if(borrado){
        sesion.setAttribute("mensaje", "Proyecto "+ idProyecto +" eliminado con éxito.");
    }else{
        sesion.setAttribute("mensaje", "ERROR: No se ha podido eliminar el proyecto "+ idProyecto);
    }

    siguientePagina = MOSTRAR_PROYECTOS;
}
```

Figura 19. Controlador captura acción eliminar proyecto

Editar

Para editar un proyecto se ha utilizado el método modificarProyecto que recibe como parámetros los nuevos valores del proyecto a modificar. Se realiza un UPDATE en la base de datos de los valores que se desean modificar del proyecto identificado por su id.

```
public boolean modificarProyecto(int idProyecto, int idEmpresa) {
    boolean hecho = false;
    Log.logBd.info("CONSULTA ModificarProyecto");
    try {
        conexion = ConexionBd.getConexion();
        Log.logBd.info("Realizada conexion - modificarProyecto()");
        Statement s = conexion.createStatement();
        ResultSet resultado = s.executeQuery("select Empresa_IdEmpresa from proyecto where IdProyecto=" + idProyecto + ";");

        //Hacemos una lista con los id de las empresas para comprobar que se elige una empresa que esta en la BD
        Statement s1 = conexion.createStatement();
        ResultSet resultado1 = s1.executeQuery("select Empresa_IdEmpresa from proyecto");
        List<Integer> lista = new ArrayList<>();
        while (resultado1.next()) {
            lista.add(resultado1.getInt("Empresa_IdEmpresa"));
        }

        //Solo si la empresa a la que se quiere cambiar el proyecto es la diferente a la que esta el proyecto se modifica
        if (resultado.next()) {
            if (resultado.getInt("Empresa_IdEmpresa") != idEmpresa && lista.contains(idEmpresa)) {
                int codigo = s.executeUpdate("update proyecto set Empresa_IdEmpresa=" + idEmpresa + " where IdProyecto=" + idProy
                //Si la consulta se ha realizado correctamente
                if (codigo > 0) {
                    //Borramos las entradas de ese proyecto en la tabla proyecto empleado
                    codigo = s.executeUpdate("delete from proyecto_empleado where proyecto_id_proyecto=" + idProyecto);
                    if (codigo > 0) {
                        Log.logBd.info("Realizada consulta - modificarProyecto()");
                    } else {
                        Log.logBd.info("Consulta no ha alterado la tabla o consulta errónea - modificarProyecto() en proyecto_emp
```

Figura 20. Método modificar proyecto

El usuario selecciona un proyecto desde la vista mostrarProyectos y pulsa el botón Editar. Este botón genera una nueva vista con un formulario para introducir los nuevos datos del proyecto. El controlador captura la acción junto con los nuevos valores introducidos en el formulario y llama al método modificarProyecto pasándole dichos valores.

```
else if(accion.equalsIgnoreCase("editarProyecto")){
    int idProyecto = Integer.parseInt(request.getParameter("idProyecto"));
    int idEmpresa = Integer.parseInt(request.getParameter("idEmpresa"));

    Log.log.info("Proyecto seleccionado id("+ idEmpresa +")");

    boolean exito = consulta.modificarProyecto(idProyecto, idEmpresa);

    if(exito){
        siguientePagina = MOSTRAR_PROYECTOS;
        sesion.setAttribute("mensaje", "Proyecto "+ idProyecto +" modificado con éxito.");
    }else{
        siguientePagina = EDITAR_PROYECTO;
        sesion.setAttribute("mensaje", "ERROR: No se ha podido modificar el proyecto "+ idProyecto);
    }
}
```

Figura 21. Controlador captura acción editar proyecto

Mostrar Empresas

Se ha implementado un método para mostrar todas las empresas dadas de alta en la base de datos.

```java
public List mostrarEmpresa() {
    ArrayList<Empresa> lista_empresas = new ArrayList<>();
    Log.logBd.info("CONSULTA MostrarEmpresa");
    try {
        conexion = ConexionBd.getConexion();
        Log.logBd.info("Realizada conexion - mostrarEmpresa()");
        Statement s = conexion.createStatement();
        ResultSet resultado = s.executeQuery("select * from empresa");
        Log.logBd.info("Realizada consulta - mostrarEmpresa()");

        while (resultado.next()) {
            Empresa empresa = new Empresa();

            empresa.setCodigoPostal(resultado.getInt("CodigoPostal"));
            empresa.setCorreo(resultado.getString("Correo"));
            empresa.setDireccion(resultado.getString("Calle"));
            empresa.setIdEmpresa(resultado.getInt("IdEmpresa"));
            empresa.setNombre(resultado.getString("Nombre"));
            empresa.setTelefono(resultado.getInt("Telefono"));

            lista_empresas.add(empresa);
        }
    } catch (SQLException error) {
        Log.logBd.error("ERROR SQL en mostrarEmpresa(): " + error);
        Log.logBd.error("                                SQL State - " + error.getSQLState());
        Log.logBd.error("                                ErrorCode - " + error.getErrorCode());
    }

    Log.logBd.info("Consulta realizada con éxito - mostrarEmpresa()");
    return lista_empresas;
```

Figura 22. Método mostrar empresas

Por otro lado, tenemos una vista con un botón Mostrar Empresas que incluye una referencia a otra vista mostrarEmpresas.jsp.

```jsp
<div class="col-md-4 d-flex justify-content-center">
   <a href="./mostrarEmpresas.jsp"><span class="fa fa-building" style="font-size:180px;color: #bd043b;text-align:cent
</div>
```

Posteriormente, se abre una nueva vista con la lista de empresas. En esta vista representada mediante un jsp llamamos al método mostrarEmpresas.

```jsp
   <%
       ConsultaBd empresa=new ConsultaBd();
       List<Empresa> lista_empresas=empresa.mostrarEmpresa(); //lista de todas las empresas
       Iterator<Empresa> iterador=lista_empresas.iterator();
       Empresa e=null;
       while(iterador.hasNext()){ //recorre la lista de empresas
           e=iterador.next();
   %>
   <tbody>
       <tr>
           <td>
             <input type="radio" id="empresa" name="empresa" value="<%= e.getIdEmpresa() %>" required>
           </td>
           <td><%= e.getIdEmpresa() %></td>
           <td><%= e.getNombre() %></td>
           <td><%= e.getDireccion() %></td>
           <td><%= e.getCodigoPostal() %></td>
           <td><%= e.getCorreo() %></td>
           <td><%= e.getTelefono() %></td>
       </tr>
       <% } %>
   </tbody>
/table>
div class="col-md-12 text-right">
   <button type="submit" class="btn btn-danger text-right" style="height:40px">
       Editar
   </button>
   <button type="submit" formaction="MainController?action=elegirEmpresa&boton=eliminar" class="btn btn-danger
       Eliminar
```

Figura 23. Mostrar empresas desde la vista

En esta misma vista, el usuario tiene opciones de añadir una nueva empresa, eliminarla o editarla.

Añadir

Para añadir empresas se ha implementado el método anadirEmpresa. Este método se basa en hacer un insert en la base de datos de un objeto de tipo empresa con unos atributos definidos previamente.

```
public boolean anadirEmpresa(Empresa empresa) {
    Log.logBd.info("CONSULTA AnadirEmpresa");
    Boolean exito = false;
    try {
        conexion = ConexionBd.getConexion();
        Log.logBd.info("Realizada conexion - anadirEmpresa()");
        Statement s = conexion.createStatement();

        int codigo = s.executeUpdate("INSERT INTO empresa(IdEmpresa, Nombre, Calle, CodigoPostal, Correo, Telefono) VALUES (" +
                + "','" + empresa.getDireccion() + "'," + empresa.getCodigoPostal() + ",'" + empresa.getCorreo() + "'," + empres
        if (codigo > 0) {
            exito = true;
            Log.logBd.info("Realizada consulta - anadirEmpresa()");
        } else {
            Log.logBd.info("Consulta no ha alterado la tabla o consulta errónea - anadirEmpresa()");
        }

    } catch (SQLException error) {
        Log.logBd.error("ERROR SQL en darAlta(): " + error);
        Log.logBd.error("            SQL State - " + error.getSQLState());
        Log.logBd.error("            ErrorCode - " + error.getErrorCode());
    }

    return exito;
}
```

Figura 24. Método añadir empresa

Al presionar el botón añadir desde la vista anterior, el usuario obtiene un formulario donde ha de ingresar los datos de la empresa que desea dar de alta. Al presionar el botón Enviar, el controlador recibe la acción y llama al método anadirEmpresa pasándole un objeto de tipo empresa con los atributos del formulario que nuestro usuario ha enviado.

```
else if(accion.equalsIgnoreCase("anadirEmpresa")){
    Empresa empresa = new Empresa();
    int idEmpresa = Integer.parseInt(request.getParameter("idEmpresa"));
    String nombre = request.getParameter("nombre");
    empresa.setIdEmpresa(idEmpresa);
    empresa.setNombre(nombre);
    empresa.setDireccion(request.getParameter("calle"));
    empresa.setCodigoPostal(Integer.parseInt(request.getParameter("codigo")));
    empresa.setCorreo(request.getParameter("correo"));
    empresa.setTelefono(Integer.parseInt(request.getParameter("telefono")));

    boolean exito = consulta.anadirEmpresa(empresa);

    if(exito){
        siguientePagina = MOSTRAR_EMPRESAS;
        sesion.setAttribute("mensaje", "Empresa "+ idEmpresa +"-"+ nombre +" añadida con éxito");
    }else{
        siguientePagina = ANADIR_EMPRESA;
        sesion.setAttribute("mensaje", "ERROR: No se pudo modificar la empresa "+ idEmpresa +"-"+ nombre);
    }
}
```

Figura 25. Controlador captura la acción añadir empresa

Eliminar

Para eliminar una empresa tenemos el método borrarEmpresa(id) que realiza una consulta en la base de datos para eliminar todos los registros de las tablas que dependen de la empresa identificada por el id. Posteriormente, se realiza otra consulta para eliminar la empresa

indicada.

```
public boolean borrarEmpresa(int idEmpresa) {
    Log.logBd.info("CONSULTA BorrarEmpresa");

    boolean hecho = true;
    try {
        conexion = ConexionBd.getConexion();
        Log.logBd.info("Realizada conexion - borrarEmpresa()");
        Statement s = conexion.createStatement();

        Log.logBd.info("Dando de baja los empleados de la empresa id(" + idEmpresa + ")");
        ResultSet empleados = getEmpleadoEmpresa(idEmpresa);
        //Borramos a los usuarios de la empresa
        while (empleados.next()) {
            String correo = empleados.getString("Correo");
            boolean borrado = darBaja(correo);
            if (!borrado) {
                hecho = false;
            } else {
                Log.logBd.error("No se ha podido borrar el empleado correo(" + correo + ")");
            }
        }

        Log.logBd.info("Borrando los proyectos de la empresa id(" + idEmpresa + ")");
        ResultSet proyectos = s.executeQuery("select IdProyecto from proyecto where Empresa_IdEmpresa=" + idEmpresa + ";");
        //Borramos cada uno de los proyectos
        while (proyectos.next()) {
            int idProyecto = proyectos.getInt("IdProyecto");
            boolean borrado = borrarProyecto(idProyecto);
            if (!borrado) {
                hecho = false;
```

Figura 26. Método borrar empresa

El usuario selecciona una empresa a eliminar desde la vista y, esta acción, se pasa al controlador. El controlador obtiene de la vista el identificador de la empresa a eliminar y realiza una llamada al método borrarEmpresa pasándole como párametro el id obtenido.

```
else if(accion.equalsIgnoreCase("eliminarEmpresa")){
    int idEmpresa = Integer.parseInt(request.getParameter("idEmpresa"));

    boolean borrado = consulta.borrarEmpresa(idEmpresa);

    if(borrado){
        sesion.setAttribute("mensaje", "Empresa "+ idEmpresa +" eliminada con éxito.");
    }else{
        sesion.setAttribute("mensaje", "ERROR: No se ha podido eliminar la empresa "+ idEmpresa);
    }

    siguientePagina = MOSTRAR_EMPRESAS;
}
```

Figura 26. Controlador captura la acción eliminar empresa

Editar

Para editar una empresa se ha utilizado el método modificarEmpresa que recibe como parámetros los nuevos valores de la empresa a modificar. Se realiza un UPDATE en la base de datos de los valores que se desean modificar de la empresa identificada por su id.

```
public boolean modificarEmpresa(int idEmpresa, String nombre, String calle, int codigoPostal, String correo, int telefono) {
    Log.logBd.info("CONSULTA ModificarEmpresa");
    boolean hecho = false;
    try {
        conexion = ConexionBd.getConexion();
        Log.logBd.info("Realizada conexion - modificarEmpresa()");
        Statement s = conexion.createStatement();
        int codigo = s.executeUpdate("UPDATE empresa set Nombre ='" + nombre + "', Calle='" + calle + "', CodigoPostal=" + codig
                + ", Correo='" + correo + "', Telefono=" + telefono + " where IdEmpresa=" + idEmpresa + ";");

        //Si la consulta se ha realizado correctamente
        if (codigo > 0) {
            hecho = true;
            Log.logBd.info("Realizada consulta - modificarEmpresa()");
        } else {
            Log.logBd.info("Consulta no ha alterado la tabla o consulta errónea - modificarEmpresa() - cod." + codigo);
        }
    } catch (SQLException error) {
        Log.logBd.error("ERROR SQL en modificarEmpresa(): " + error);
        Log.logBd.error("                                  SQL State - " + error.getSQLState());
        Log.logBd.error("                                  ErrorCode - " + error.getErrorCode());
    }

    return hecho;
}
```

Figura 27. Método modificar empresa

El usuario selecciona una empresa desde la vista mostrarEmpresas y pulsa el botón Editar. Este botón genera una nueva vista con un formulario para introducir los nuevos datos de la empresa. El controlador captura la acción junto con los nuevos valores introducidos en el formulario y llama al método modificarEmpresa pasándole dichos valores.

```
else if(accion.equalsIgnoreCase("editarEmpresa")){
    int idEmpresa = Integer.parseInt(request.getParameter("idEmpresa"));
    String nombre = request.getParameter("nombre");
    String calle = request.getParameter("calle");
    int codigoPostal = Integer.parseInt(request.getParameter("codigo"));
    String correo = request.getParameter("correo");
    int telefono = Integer.parseInt(request.getParameter("telefono"));

    boolean exito = consulta.modificarEmpresa(idEmpresa, nombre, calle, codigoPostal, correo, telefono);

    if(exito){
        siguientePagina = MOSTRAR_EMPRESAS;
        sesion.setAttribute("mensaje", "Empresa "+ idEmpresa +"-"+ nombre +" modificada con éxito");
    }else{
        siguientePagina = EDITAR_EMPRESA;
        sesion.setAttribute("mensaje", "ERROR: No se pudo modificar la empresa "+ idEmpresa +"-"+ nombre);
    }
}
```

Figura 28. Controlador captura la acción editar empresa

Mostrar Peticiones

Se ha implementado un método para mostrar todas las peticiones realizadas por los empleados para solicitar días libres.

```java
public List mostrarSolicitudes() {
    ArrayList<Solicitud> lista_solicitudes = new ArrayList<>();
    Log.logBd.info("CONSULTA MostrarSolicitudes");
    try {
        conexion = ConexionBd.getConexion();
        Log.logBd.info("Realizada conexion - mostrarSolicitudes()");
        Statement s = conexion.createStatement();
        ResultSet resultado = s.executeQuery("select * from diaLibre");
        Log.logBd.info("Realizada consulta - mostrarSolicitudes()");

        while (resultado.next()) {
            Solicitud solicitud = new Solicitud();
            solicitud.setFechaFin(resultado.getDate("FechaFin"));
            solicitud.setFechaInicio(resultado.getDate("FechaInicio"));
            solicitud.setMotivo(resultado.getString("Motivo"));
            solicitud.setAprobado(resultado.getBoolean("Aprobado"));
            solicitud.setLeido(resultado.getBoolean("Leido"));
            solicitud.setTramitado(resultado.getBoolean("Tramitado"));
            solicitud.setUsuario(getUsuario(resultado.getString("EmpleadoEmpresa_Correo")));

            lista_solicitudes.add(solicitud);
        }
    } catch (SQLException error) {
        Log.logBd.error("ERROR SQL en mostrarSolicitudes(): " + error);
        Log.logBd.error("                    SQL State - " + error.getSQLState());
        Log.logBd.error("                    ErrorCode - " + error.getErrorCode());
    }

    Log.logBd.info("Consulta realizada con éxito - mostrarSolicitudes()");
    return lista_solicitudes;
```

Figura 29. Método mostrar solicitudes

Por otro lado, tenemos una vista con un botón Peticiones que incluye una referencia a otra vista solicitudesPendientes.jsp.

```jsp
<li class="nav-item">
    <a class="nav-link" href="solicitudesPendientes.jsp">Peticiones</a>
</li>
```

Figura 30. Referencia al jsp solicitudes pendientes

Posteriormente, se abre una nueva vista con la lista de solicitudes. En esta vista representada mediante un jsp llamamos al método mostrarSolicitudes

```jsp
<br><br>
<%
        ConsultaBd consulta =new ConsultaBd();
        List<Solicitud> solicitudes_pendientes = consulta.mostrarSolicitudes(); //lista de todas las solicitudes
        Iterator<Solicitud> iterador= solicitudes_pendientes.iterator();
         Solicitud solicitud =null;
        while(iterador.hasNext()){ //recorre la lista de solicitudes
            solicitud =iterador.next();
            if(!solicitud.isTramitado()) //si la solicitud no se ha tramitado todavia la mostramos
            {

%>
<div class="shadow p-3 mb-5 bg-white rounded">
    <p><%= solicitud.getUsuario().getIdUsuario() %> - <%= solicitud.getUsuario().getNombre() %></p>
    <p>  <%= solicitud.getFechaInicio() %> - <%= solicitud.getFechaFin() %> </p>
    <p>Motivo: <%= solicitud.getMotivo() %></p>

    <div class="d-flex flex-row-reverse">
        <a href="CalendarController?action=Aprobar&fecha_i=<%= solicitud.getFechaInicio() %>&fecha_f=<%= solicitud.getFecha
        <button type="button" class="btn btn-success btn-circle btn-sm" style="margin:5px;">Aprobar</button>
        </a>
        <a href="CalendarController?action=Denegar&fecha_i=<%= solicitud.getFechaInicio() %>&fecha_f=<%= solicitud.getFecha
            <button type="button" class="btn btn-danger btn-circle btn-sm" style="margin:5px;">Rechazar</button>
        </a>
    </div>
</div>
<% }} %>
```

Figura 31. Mostrar la solicitud desde la vista

En esta misma vista, el usuario tiene opciones de aprobar o denegar solicitudes.

Aprobar/Denegar

Estas peticiones comentadas anteriormente pueden ser aprobadas o rechazadas por el personal de recursos humanos. Para ello, se ha implementado un método aprobarSolicitud con unos parámetros para indicar si la solicitud se desea aprobar o rechazar. Al llamar al método aprobarSolicitud desde el controlador, la solicitud especificada cambia su estado a aprobado o denegado.

```
public boolean aprobarSolicitud(boolean aprobada, boolean leida, boolean tramitada, Date fecha_i, Date fecha_
    Log.logBd.info("CONSULTA Aprobar Solicitud");
    boolean hecho = false;
    int cod;

    try {
        conexion = ConexionBd.getConexion();
        Log.logBd.info("Realizada conexion - aprobarSolicitud()");

        Statement s3 = conexion.createStatement();

        cod = s3.executeUpdate("UPDATE DiaLibre SET Leido=" + leida + ", Aprobado=" + aprobada + ", Tramitadc

        if (cod > 0) {

            hecho = true;
        }

    } catch (SQLException error) {
        Log.logBd.error("ERROR SQL en ficharEmpleado(): " + error);
        Log.logBd.error("                          SQL State - " + error.getSQLState());
        Log.logBd.error("                          ErrorCode - " + error.getErrorCode());
    }
    return hecho;

}
```

Figura 32. Método aprobar solicitud

Solicitar Informe

Por último, los empleados de recursos humanos tienen la opción de solicitar el informe para un período de tiempo seleccionado. Este informe puede ser por empleado, empresa o proyecto. Para ello, se ha creado un método para cada entidad: InformeEmpresa indicando el id de la empresa, InformeEmpleado indicando el id del empleado y InformeProyecto indicando el id del proyecto.

```
public ArrayList informeEmpresa(int idEmpresa, Date FechaI, Date FechaF) {
    ArrayList<String> informe = new ArrayList<>();

    try {
        conexion = ConexionBd.getConexion();
        Log.logBd.info("Realizada conexion - informeEmpresa()");

        Statement s3 = conexion.createStatement();
        ResultSet resultado = s3.executeQuery("select * from empresa inner join proyecto on empresa.IdEmpresa=pr

        while (resultado.next()) {

            informe.add("El día " + resultado.getString("Fecha") + " el empleado " + resultado.getString("Calenc

        }
    } catch (SQLException error) {
        Log.logBd.error("ERROR SQL en ficharEmpleado(): " + error);
        Log.logBd.error("                          SQL State - " + error.getSQLState());
        Log.logBd.error("                          ErrorCode - " + error.getErrorCode());
    }
    return informe;
}
```

Figura 33. Método informe empresa

Implementación Empleado

Por otro lado, tenemos la implementación de las funciones que pueden realizar los empleados dentro de la aplicación. Entre ellas, podemos destacar principalmente fichar un empleado, ver el calendario del empleado o solicitar un día libre.

Fichar

Para fichar un empleado se ha implementado un método ficharEmpleado pasándole como parámetros la fecha en la que se produce la acción, la hora de entrada y la hora de salida, así como el correo del empleado y el proyecto en el que ha trabajado.

```
public boolean ficharEmpleado(Date fecha, Time hora_entrada, Time hora_salida, String correo, int id_proyecto) t
    boolean hecho = false;
    int cod = 0;

    try {
        conexion = ConexionBd.getConexion();
        Log.logBd.info("Realizada conexion - ficharEmpleado()");
        Statement s = conexion.createStatement();
        int codigo = s.executeUpdate("INSERT INTO Calendario VALUES('" + fecha + "','" + hora_entrada + "','" +
        Statement s2 = conexion.createStatement();
        ResultSet resultado = s2.executeQuery("select Horas from Proyecto_Empleado where proyecto_id_proyecto="
        int horas;

        int horas_trabajadas_hoy = getHoras(hora_entrada.toString(), hora_salida.toString());
        int horas_totales = 0;
        Statement s3 = conexion.createStatement();
        while (resultado.next()) {
            horas = resultado.getInt("Horas");

            horas_totales = horas + horas_trabajadas_hoy;
            cod = s3.executeUpdate("UPDATE Proyecto_Empleado SET horas=" + horas_totales + " where proyecto_id_p

        }

        if (codigo > 0 && cod > 0) {

            hecho = true;
        }
```

Figura 34. Método fichar empleado

Tras introducir los datos en el formulario correspondiente, el empleado pulsa el botón enviar y esta acción se pasa al controlador CalendarioController. Desde ahí se obtienen los valores de los parámetros introducidos por el usuario y se llama al método descrito anteriormente para registrar esos valores en la base de datos.

```
if (accion.equalsIgnoreCase("fichar")) {
    String fecha1 = request.getParameter("hora_entrada");
    String fecha2 = request.getParameter("hora_salida");
    String idProyecto = request.getParameter("idProyecto");

    String fecha_1 = fecha1.substring(0, 10);

    String hora_entrada = fecha1.substring(11) + ":00";
    String hora_salida = fecha2.substring(11) + ":00";
    Usuario usuario = (Usuario) sesion.getAttribute("usuarioSesion");

    try {
        boolean hecho = consulta.ficharEmpleado(Date.valueOf(fecha_1), Time.valueOf(hora_entrada), Time.valu
        if (hecho) {
            siguientePagina = "inicioUser.jsp";
            sesion.setAttribute("mensaje", "Operación realizada con éxito.");
        } else {
            siguientePagina = "fichar.jsp";
            sesion.setAttribute("mensaje", "ERROR: No se ha podido realizar la operación");
        }

    } catch (ParseException ex) {
        Logger.getLogger(CalendarController.class.getName()).log(Level.SEVERE, null, ex);
    }
```

Figura 35. Controlador captura la acción fichar empleado

Calendario

El empleado tiene la opción de poder visualizar el calendario con sus días libres asignados. Esta acción es posible gracias al método getDiasLibres(correo) que devuelve todos los registros de los días libres aprobados que hay en la base de datos del empleado identificado por el correo.

```
public List getDiasLibres(String correo) {
    ArrayList<Solicitud> lista_diasLibres = new ArrayList<>();
    Log.logBd.info("CONSULTA getDiasLibres");
    try {
        conexion = ConexionBd.getConexion();
        Log.logBd.info("Realizada conexion - getDiasLibres()");
        Statement s = conexion.createStatement();
        ResultSet resultado = s.executeQuery("select * from DiaLibre where EmpleadoEmpresa_Correo ='" + correo
        Log.logBd.info("Realizada consulta - getDiasLibres()");

        while (resultado.next()) {
            Solicitud solicitud = new Solicitud();
            solicitud.setFechaFin(resultado.getDate("FechaFin"));
            solicitud.setFechaInicio(resultado.getDate("FechaInicio"));
            solicitud.setMotivo(resultado.getString("Motivo"));
            solicitud.setAprobado(resultado.getBoolean("Aprobado"));
            solicitud.setLeido(resultado.getBoolean("Leido"));
            solicitud.setTramitado(resultado.getBoolean("Tramitado"));
            solicitud.setUsuario(getUsuario(resultado.getString("EmpleadoEmpresa_Correo")));

            lista_diasLibres.add(solicitud);
        }
    } catch (SQLException error) {
        Log.logBd.error("ERROR SQL en getDiasLibres(): " + error);
        Log.logBd.error("                    SQL State - " + error.getSQLState());
        Log.logBd.error("                    ErrorCode - " + error.getErrorCode());
    }

    Log.logBd.info("Consulta realizada con éxito - getDiasLibres()");
```

Figura 36. Método obtener días libres

Además, también se pueden visualizar los días laborales del empleado a través de un método getCalendario(correo) que contiene el registro de los días que el empleado ha estado trabajando.

```
public List getCalendario(String correo) {
    ArrayList<Jornada> lista_jornadas = new ArrayList<>();
    Log.logBd.info("CONSULTA getCalendario");
    try {
        conexion = ConexionBd.getConexion();
        Log.logBd.info("Realizada conexion - getCalendario()");
        Statement s = conexion.createStatement();
        ResultSet resultado = s.executeQuery("select * from Calendario where Correo ='" + correo + "';");
        Log.logBd.info("Realizada consulta - getCalendario()");

        while (resultado.next()) {
            Jornada jornada = new Jornada();
            jornada.setFecha(resultado.getDate("Fecha"));
            jornada.setHora_entrada(resultado.getTime("HoraEntrada"));
            jornada.setHora_salida(resultado.getTime("HoraSalida"));
            jornada.setCorreo(resultado.getString("Correo"));
            jornada.setId_proyecto(resultado.getInt("id_proyecto"));

            lista_jornadas.add(jornada);
        }
    } catch (SQLException error) {
        Log.logBd.error("ERROR SQL en getCalendario(): " + error);
        Log.logBd.error("                               SQL State - " + error.getSQLState());
        Log.logBd.error("                               ErrorCode - " + error.getErrorCode());
    }

    Log.logBd.info("Consulta realizada con éxito - getCalendario()");
    return lista_jornadas;
}
```

Figura 37. Método obtener calendario

Este método se llama posteriormente en la vista para mostrar los registros comentados anteriormente.

Solicitar Día Libre

Por último, el empleado podrá solicitar un día libre o varios indicando la fecha de inicio y fin, así como el motivo. Si solo solicita un día libre la fecha final es la misma que la fecha inicial.

Para realizar esta acción se ha implementado el método solicitarDíaLibre que se llamará posteriormente desde el controlador CalendarioController.

```
public boolean solicitarDiaLibre(Date fechaI, Date fechaF, String motivo, String correo) {
    boolean hecho = false;

    try {
        conexion = ConexionBd.getConexion();
        Log.logBd.info("Realizada conexion - solicitarDiaLibre()");
        Statement s = conexion.createStatement();
        int codigo = s.executeUpdate("INSERT INTO DiaLibre VALUES('" + fechaI + "','" + fechaF + "','" + motivo

        if (codigo > 0) {

            hecho = true;
        }

    } catch (SQLException error) {
        Log.logBd.error("ERROR SQL en solicitarDiaLibre(): " + error);
        Log.logBd.error("                          SQL State - " + error.getSQLState());
        Log.logBd.error("                          ErrorCode - " + error.getErrorCode());
    }
    return hecho;

}
```

Figura 38. Método solicitar día libre

BIBLIOGRAFÍA

[1] https://www.cssscript.com/create-simple-event-calendar-javascript-caleandar-js/

[2] https://www.w3schools.com/js/js_errors.asp

[3] https://danlaho.wordpress.com/2010/05/04/login-de-usuario-jspservlet/

[4] https://www.arquitecturajava.com/usando-java-session-en-aplicaciones-web/

[5] https://tomcat.apache.org/tomcat-5.5-doc/servletapi/javax/servlet/http/HttpSession.html

[6] https://tutobasico.com/instalar-tomcat-windows/

[7] https://support.academicsoftware.eu/hc/es/articles/360007014958-C%C3%B3mo-instalar-MySQL-Workbench#:~:text=Paso%201%3A%20Ve%20a%20la,Next%20para%20iniciar%20la%20instalaci%C3%B3n.

[8] https://docs.oracle.com/javase/7/docs/api/java/util/logging/Logger.html

[9] https://codigosparadesarrolladores.blogspot.com/2014/04/codigo-java-conectar-nuestra-aplicacion-con-una-base-de-datos.html

[10] https://developer.mozilla.org/es/docs/Web/API/Element/getAttribute