Proyecto NoSQL

White Group

aplicación

Se desarrolla una parte de una aplicación más grande a la que no tenemos acceso que se encargará del mantenimiento de la tabla Alumnos:

- Create: alta de nuevos alumnos
- Read: recuperar la información de la base de datos
- Update: modificación de los alumnos existentes
- Delete: eliminación de los registros

aplicación

Características técnicas destacables:

- Uso BDD PostgreSQL, aprovechando características NOSQL (JSON)
- Uso tecnologías HTML + CSS + JS en frontend
- Uso tecnologías Java (JSP + Servlet) en el backend
- Servidor GlassFish

aplicación

Detalles destacables de implementación:

- Interfaz consola
- Interfaz WEB
- Escalabilidad sin recompilación (en los atributos de los alumnos)

aplicación

Carencias destacables de implementación:

- Cambio de paradigma según el planteamiento inicial (AJAX-JSON)
- Comprobación datos (condiciones ideales, sin errores)
- mecanismos para evitar mal uso de la interfaz web

Base de Datos PostgreSQL

tecnología de la base de datos.

- Tiene muchas cualidades que lo hacen ser una muy buena alternativa para instalar sistemas en empresas, universidades y una gran cantidad de otras aplicaciones.

- Es un avanzado sistema de bases de datos relacionales Open Source, es decir, que cualquier persona puede colaborar en su desarrollo y modificación.

Base de Datos PostgreSQL

- Una sesión en PostgreSQL consiste en la ejecución de el servidor y de la aplicación cliente:
    - El servidor es el que maneja archivos de bases de datos, acepta conexiones a las aplicaciones cliente y actúa en la base de datos.
    - La aplicación cliente es la que realiza operaciones en la base de datos. Estas pueden ser aplicaciones de texto en consola, gráficas, un servidor web que acceda a la base de datos para mostrar una página o herramientas especializadas para el manejo de bases de datos.

Ventajas:

- **Open Source, Multiplataforma, Alto volumen (Big-Data), Seguridad de la información, Facilidad de manejo.**

frontend HTML+CSS

El diseño de nuestro proyecto, está basado en los diseños originales de PostgreSQL.

Tipografía, colores y formas.

Manteniendo el aspecto lo más fiel a su original.

frontend HTML+CSS

Es un formato en HTML5 + CSS,

dividido en tres páginas.

- Página principal
- Visionado de la base de datos.
- Modificación de datos.

La primera nos permite el acceso a la BBDD.

En esta segunda, podemos ver su contenido

via HTML y crear, eliminar o modificar datos.

frontend HTML+CSS

El siguiente formulario, nos proporciona todo lo necesario para insertar un nuevo usuario.

El equipo de backend, ha tenido en cuenta, que el cliente, puede modificar los campos de características, sin tener que modificar nada de programación, debiendo como mucho, modificar parte del cuerpo de HTML.

frontend HTML+CSS

En cuanto al CSS, la novedad está en el uso de “display GRID”.

Y ordenar todo el contenido dentro de cajas que hemos introducido en un mismo contenedor, a efecto que todo el contenido quedara lo más centrado posible.

```
table, th, td{
    padding-right: 10px;
    padding-left: 10px;
    margin: 10px;
    text-align: left;

}
.iconos {
    text-align: center;

}
.contenedor {

    justify-content: space-around;
    display: grid;
    grid-template-columns: auto;
    grid-gap: 5px;
    padding: 1%;
    margin: 0%;
}

.contenedor > div {
  border: 2px solid #909090;
  border-radius: 12px;
  font-size: 0.9em;
}
```

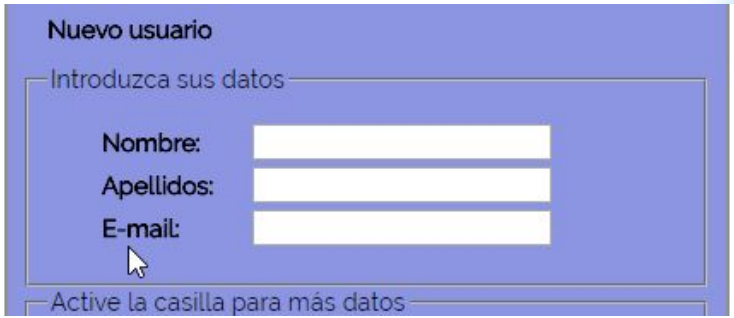
frontend HTML+CSS

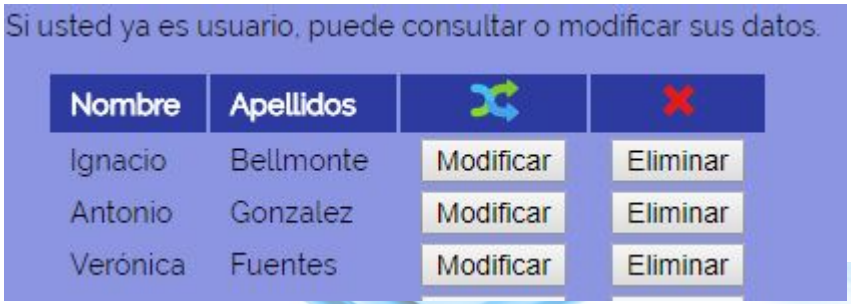
Para alinear las cajas de texto,

el recurso más socorrido ha sido

el uso de una tabla, que después

hemos formateado con un poco de CSS.






```
20  }
21  table, th, td{
22      padding-right: 10px;
23      padding-left: 10px;
24      margin: 10px;
25      text-align: left;
26
27  }
```

frontend JS

Para añadir pequeñas funcionalidades en el frontend, hemos usado JavaScript "Vanilla"

Utilizamos 3 métodos y el evento onload:

1. `Function capturaEventoClick()`
   - En esta función capturamos el evento "CLICK" de forma que siempre sabremos si la casilla del checkbox está seleccionada o no para poder actuar dependiendo del caso.

frontend JS

2. `Function desactivaCamposInput()`

- Con este método activaremos o desactivaremos campos de texto en función de los campos checkbox y del contenido (en caso de edición de un Alumno). Es decir que si el checkbox está seleccionado podremos escribir texto y si lo desactivamos, el campo de texto siempre quedará vacío y desactivado.

frontend JS

3. `Function asignarEventosCamposCheck()`

En la tercera y última función:

- Creamos una variable llamada “elementos” en la que guardamos todos los selectores de tipo “input type= ‘checkbox’”.
- Esto nos dará un ARRAY que recorremos para asignar a cada uno de los “checkbox” el evento CLICK.

frontend JS

4.
```
window.onload = function(){
    desactivaCamposInput();
    asignarEventoClickcamposCheck();
 }
```

Con el evento onload hacemos que al cargar la página jsp , en nuestro caso sería editar.jsp, el javaScript se iniciará gracias al evento “onload” y llamará a las funciones creadas con anterioridad (desactivaCamposInput() y asignarEventoClickcamposCheck();)

detalle HTML escalabilidad

la implementación actual nos permite añadir nuevos atributos de usuario sin tener que recompilar el código Java

simplemente repitiendo este trozo de código (y cambiando los nombres de los *id* y algún *value*

```
<tr>
    <th><input id="check1" type="checkbox" name="check1" value="1" class="js-check">
        <label for="texto1">Sexo: </label>
    </th>
    <th>
        <input id="texto1" type="text" maxlength="20" size="20" name="valor" class="js-input">
        <input id="hidden1" type="hidden" name="clave" value="sexo" class="js-input">
    </th>
</tr>
```

detalle HTML escalabilidad

el javascript está diseñado para encontrar todos los elementos que cumplan los criterios, haciendo que las modificaciones en el HTML (añadir un nuevo atributo) se vean afectadas

```
var elementos = document.querySelectorAll("input[type='checkbox']");
```

```
var elementos = document.querySelectorAll("input.js-input");
```

detalle HTML escalabilidad

el java en el jsp/servlet también está pensado para capturar todos esos datos:

```
String name = request.getParameter("nombre");
String ape1 = request.getParameter("ape1");
String mail = request.getParameter("email");
String[] valor = request.getParameterValues("valor");
String[] clave = request.getParameterValues("clave");
```

detalle HTML escalabilidad

como estos datos se guardan en un campo JSON de la base de datos, no hace falta modificar la estructura de la tabla

	id integer	nombre text	apellido text	mail text	caracteristicas jsonb
1	15	Luis	Vecino	veci...	{"ojos":"marrones","pelo":"calvo","nacionalidad":"gallego"}
2	16	Daniel	Pare	pare...	{"ojos":"marrones","pelo":"castaño","nacionalidad":"cat"}
3	17	Ricart	Valira	valir...	{"ojos":"verdes","pelo":"canoso","nacionalidad":"cat"}
4	18	Miguel	Mate	mat...	{"ojos":"celestes","pelo":"moreno","nacionalidad":"ovni"}

java

trabajado dividido en 2 partes:

- jsp + servlets que interaccionan con el frontend HTML
- clases que gestionan los Alumnos y su persistencia en la base de datos

librerías java backend

2 librerías externas:

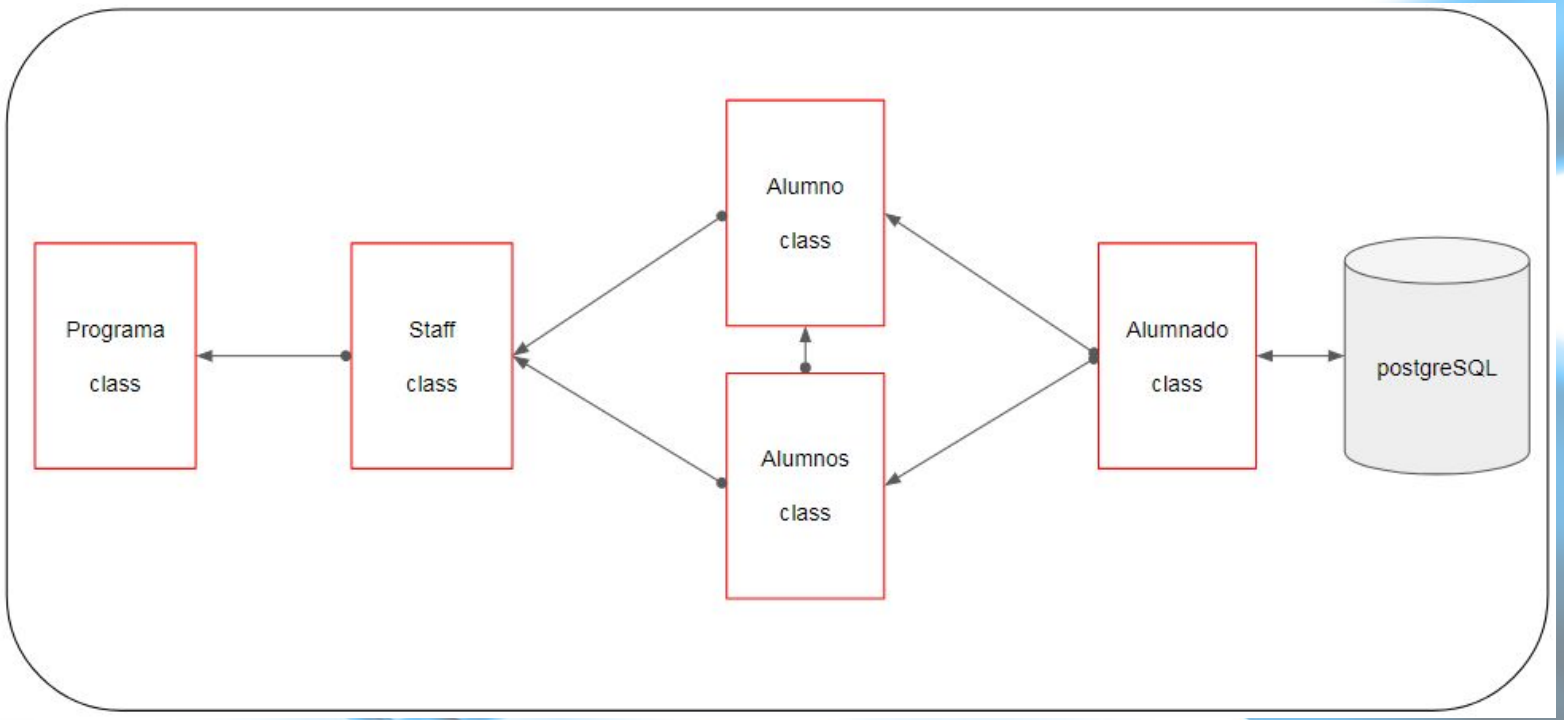
- postgresql-42.2.2.jar : conexión BDD
- json-simple-1.1.jar : manipulación JSON

jsp+servlet backend

desde jsp conseguimos añadir código de programación java dentro de una página HTML. Ejemplo:

```
<%@page import = "alumno.Alumno" %>

<%@page contentType="text/html" pageEncoding="UTF-8"%>

<%

    Alumnos alumnos = new Alumnos();

    Alumno[] alum = alumnos.listado();

%>
```

clases backend



clases backend

detalle uso librería externa para recuperar datos de JSON

```
98
99          JSONParser parser = new JSONParser();
100         JSONObject jSonObj = new JSONObject();
101
```

```
{"ojos":"celestes","pelo":"moreno","nacionalidad":"ovni"}
```

```
102         try {
103             jSonObj = (JSONObject) parser.parse(cadenaAtributos);
104         } catch (ParseException e) {
105             e.printStackTrace();
106         }
107
108         for (Object obj : jSonObj.keySet() ) {
109             String clave = (String) obj;
110             String valor = (String) jSonObj.get(clave);
111             //  System.out.println(clave + " - " + valor);
112             retorno.put(clave, valor);
113         }
```

demo consola

```
1. Listar alumnos
2. Insertar Alumno
3. Actualizar Alumno
4. Borrar Alumno
5. Detalle Alumno
6. Búsqueda

0. Finalizar el programa

Escoge una opción (1-6,0):
```

```
Listado de alumnos
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

[15] Luis Vecino
[16] Daniel Pare
[17] Ricart Valira
[18] Miguel Mate
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
```

```
Detalle de alumno
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

Indica el ID del alumno a detallar:
15
Luis Vecino vecino@luis.gll {"ojos":"marrones","pelo":"calvo","nacionalidad":"gallego"}
```

```
Búsqueda de alumnos por atributo
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

Campo:
ojos
Valor:
marrones
[15] Luis Vecino
[16] Daniel Pare
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
```

```
Búsqueda de alumnos por atributo
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

Campo:
pelo
Valor:
calvo
[15] Luis Vecino
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
```

github

https://github.com/matebcn/proyecto-nosql.git