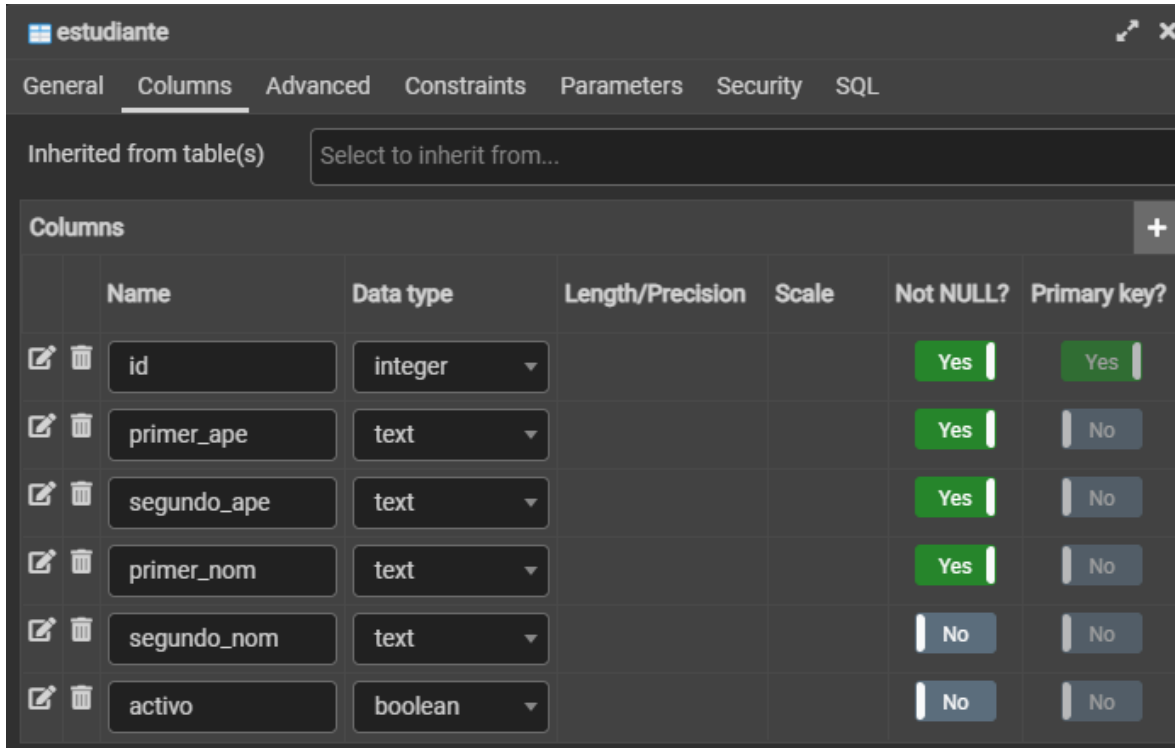


Reporte CRUD

Tomé como base el proyecto anterior, de la conexión a base de datos, y trabajé sobre él. Ya teniendo la conexión a base de datos funcionando, decidí crear una nueva base de datos en Posgre para esta tarea. Nombré la base de datos como “SistemaABC” y creé una tabla “estudiantes” con los siguientes datos:



estudiante						
General Columns Advanced Constraints Parameters Security SQL						
Inherited from table(s) Select to inherit from...						
Columns +						
	Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?
	id	integer			Yes	Yes
	primer_ape	text			Yes	No
	segundo_ape	text			Yes	No
	primer_nom	text			Yes	No
	segundo_nom	text			No	No
	activo	boolean			No	No

Imagen 1:Tabla y tipos de datos en PostgreSQL

Así también decidí usar una herramienta online (www.mockaroo.com) para crear un set de datos falsos que pudiese usar como placeholder.

Para realizar este proyecto decidí utilizar un modelo en 3 capas, donde separo la vista, los controladores y el acceso a datos. Por lo que creé un POJO (Plain Old Java Object) por medio del cual pueda representar en nuestro programa los datos recibidos del DAO (Data Acces Object) y la base de datos.

En el el DAO consideré solo 3 acciones que realizaría el sistema:

- Solicitar una lista de todos los estudiantes activos.
- Agregar un estudiante activo
- Cambiar el estado de un estudiante (“Activo” a “no activo”)

Para el caso de “Consultar estudiante” no consideré una propia acción, puesto que puede ser obtenido a partir de la lista anterior.

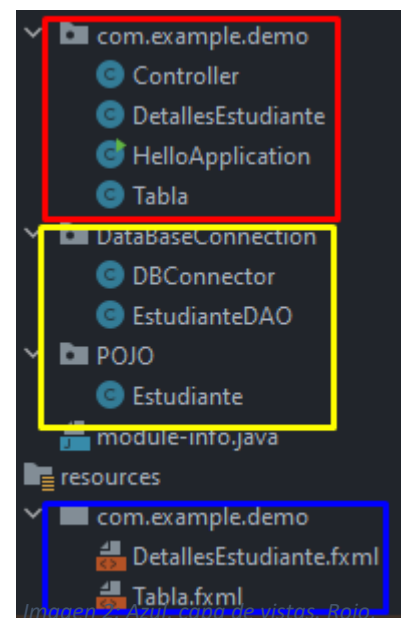


Imagen 2: Azul, capa de vistas. Rojo, lógica. Amarillo, acceso a datos.

```

1 package DataBaseConnection;
2
3 import POJO.Estudiante;
4
5 import java.sql.Connection;
6 import java.sql.PreparedStatement;
7 import java.sql.ResultSet;
8 import java.sql.SQLException;
9 import java.util.ArrayList;
10 import java.util.List;
11
12 public class EstudianteDAO {
13     @ public static List<Estudiante> getEstudiantesActivos(){...}
37
38     @ public static void guardar(Estudiante estudiante){...}
54
55     @ public static void borrar(Estudiante estudiante){...}
68 }

```

Imagen 3: Código parcial que muestra funciones del DAO de Estudiante.

Continué con la vista para nuestro programa principal. Ya que como librería gráfica decidí usar JavaFX, esta está diseñada en archivos FXML. El resultado final fue el siguiente:

Apellido Paterno	Apellido Materno	Primer Nombre	Segundo Nombre
Tabla sin contenido			

Agregar

Consultar

Borrar

Imagen 4: Vista FXML de la pantalla principal

Ahora llegaba la parte donde se dota de funcionalidad a la vista. Por medio de la interfaz “Initializable” de Java pude cargar los datos de la base de datos a la vista antes de que esta se muestre en pantalla, dando una mejor apariencia al programa:



Apellido Paterno	Apellido Materno	Primer Nombre	Segundo Nombre
Clixby	Jermayne	Farnes	Diane-marie
Gerauld	Frederich	O'Deegan	Wheeler
Clohessy	Ulrick	Merlin	Neal
Santello	Lynn	Gebbe	Richardo
Abbay	Clareta	Gorden	Aretha
Condie	Keefer	Dinse	Merrielle
Vedstra	Garth	Graffin	Niles
Choldcroft	Ariana	Moyne	Arie
Wellwood	Dotti	Edmund	Reginauld
Rosel	Stu	Mynard	Keelia
Crain	Ondrea	Rubinov	Walther

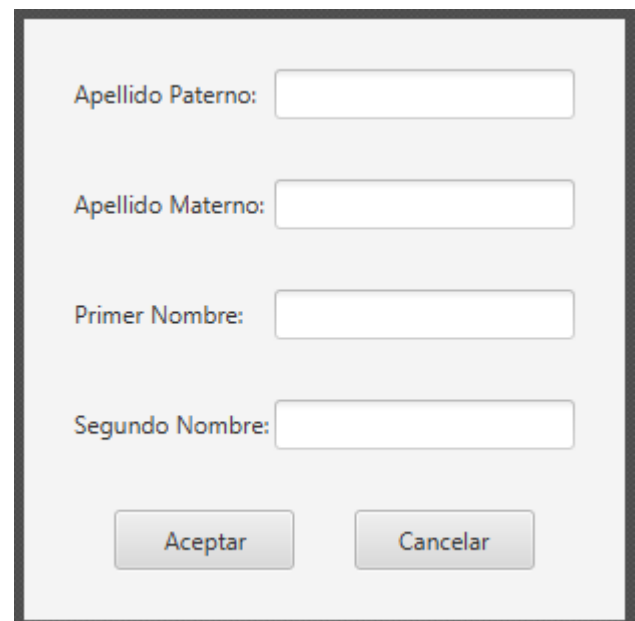
Below the table are three buttons: **Agregar**, **Consultar**, and **Borrar**.

Imagen 5: Interfaz principal con información de la base de datos.

En este punto solo restaba añadir la funcionalidad al resto de los botones. Para este caso diseñé una nueva interfaz que pude reutilizar en los escenarios de “Agregar” y “Consultar”.

Ambas interfaces son similares, así que fue posible usarla en ambos casos con un par de modificaciones en el código.

Finalmente, testeé el código y añadí las comprobaciones y alertas necesarias para que el sistema funcionase correctamente.



The dialog box contains four text input fields with labels: **Apellido Paterno:**, **Apellido Materno:**, **Primer Nombre:**, and **Segundo Nombre:**. At the bottom are two buttons: **Aceptar** and **Cancelar**.

Imagen 6: Interfaz utilizada para los casos de "Agregar" y "Consultar".