

# Índice del Proyecto

## 1. Introducción

- 1.1 Descripción del Proyecto
- 1.2 Objetivos del Proyecto

## 2. Análisis y Diseño del Sistema

- 2.1 Modelo de Datos y Diseño de la Base de Datos
- 2.2 Diagrama de Clases

## 3. Herramientas y Tecnologías Utilizadas

- 3.1 JavaFX
- 3.2 SceneBuilder
- 3.3 MySQL
- 3.4 JDBC (Java Database Connectivity)

## 4. Desarrollo de la Interfaz Gráfica

- 4.1 Estructura General de la GUI
- 4.2 Diseño de Pantallas y Componentes en SceneBuilder
- 4.3 Navegación y Flujo de la Interfaz

## 5. Implementación de la Conexión a la Base de Datos

- 5.1 Configuración de la Base de Datos MySQL
- 5.2 Configuración de JDBC para la Conexión
- 5.3 Creación de Clases Modelo para la Interacción con la Base de Datos

## 6. Desarrollo de Funcionalidades CRUD

- 6.1 Inserción de Datos desde la Interfaz
- 6.2 Lectura de Datos y Visualización en la Tabla
- 6.3 Actualización de Datos desde la GUI
- 6.4 Eliminación de Datos y Confirmaciones de Borrado

## 7. Implementación de Filtrado y Búsqueda de Datos

- 7.1 Diseño de Formularios de Filtrado
- 7.2 Generación de Consultas SQL Dinámicas para Búsquedas
- 7.3 Visualización de Resultados Filtrados

## 8. Bibliografía

# 1. Introducción

## 1.1 Descripción del Proyecto

Este proyecto consiste en el desarrollo de una aplicación de escritorio con una **interfaz gráfica de usuario (GUI)** construida en **JavaFX** y diseñada con **SceneBuilder**, la cual estará conectada a una base de datos **MySQL** para realizar operaciones de manipulación y filtrado de datos. La aplicación permitirá a los usuarios interactuar con los datos almacenados en la base de datos de manera intuitiva, sin necesidad de conocimientos técnicos sobre SQL. Las principales funcionalidades incluyen la gestión de datos (crear, leer, actualizar y eliminar - **CRUD**) y la búsqueda o filtrado de información según criterios específicos.

La solución busca facilitar la gestión de información, optimizando procesos que tradicionalmente requerirían herramientas más complejas o acceso directo a la base de datos. Esto hace que el sistema sea accesible para usuarios no especializados, mejorando su productividad.

## 1.2 Objetivos del Proyecto

### Objetivo General:

- Diseñar y desarrollar una aplicación funcional e intuitiva que permita a los usuarios manipular y consultar datos de una base de datos MySQL mediante una interfaz gráfica interactiva creada con JavaFX y SceneBuilder.

### Objetivos Específicos:

1. Crear una interfaz gráfica amigable y funcional utilizando JavaFX y SceneBuilder, enfocada en la facilidad de uso.
2. Implementar la conexión entre la aplicación y la base de datos MySQL utilizando JDBC para garantizar la sincronización de los datos.
3. Desarrollar funcionalidades CRUD que permitan la creación, lectura, actualización y eliminación de registros desde la interfaz gráfica.
4. Diseñar un sistema de búsqueda y filtrado que facilite la consulta de datos específicos mediante criterios personalizados ingresados por los usuarios.
5. Asegurar la validez de los datos mediante validaciones de entrada y manejar errores comunes de manera eficiente.
6. Proveer una solución modular y escalable, que permita futuras extensiones y mejoras.

# 2. Análisis y Diseño del Sistema

## 2.1 Modelo de Datos y Diseño de la Base de Datos

El sistema manejará un conjunto de libros, y cada libro tendrá las siguientes propiedades: **ID, nombre, autor, precio y cantidad**. Basándonos en estos requerimientos, el modelo de datos y la estructura de la base de datos será sencillo y optimizado para estas funcionalidades. Todo esto se llevará a cabo usando XAMPP.

## Modelo de Datos:

### 1. Tabla principal: **libros**

- **ID**: Identificador único del libro (*Primary Key*, entero).
- **Título**: Nombre del libro (cadena de texto, longitud máxima de 40 caracteres).
- **Autor**: Autor del libro (cadena de texto, longitud máxima de 40 caracteres).
- **Precio**: Precio del libro (decimal con dos decimales, por ejemplo, 99.99).
- **Cantidad**: Número de unidades disponibles (entero, valores positivos).

## Esquema de la Base de Datos MySQL:

#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra	Acción
<input type="checkbox"/> 1	<b>id</b>	int(11)			No	Ninguna			Cambiar  Eliminar  Más
<input type="checkbox"/> 2	<b>nombre</b>	varchar(40)	utf8mb4_general_ci		No	Ninguna			Cambiar  Eliminar  Más
<input type="checkbox"/> 3	<b>autor</b>	varchar(40)	utf8mb4_general_ci		Sí	NULL			Cambiar  Eliminar  Más
<input type="checkbox"/> 4	<b>precio</b>	decimal(4,2)			No	Ninguna			Cambiar  Eliminar  Más
<input type="checkbox"/> 5	<b>cantidad</b>	int(11)			No	Ninguna			Cambiar  Eliminar  Más

## Explicación del Esquema:

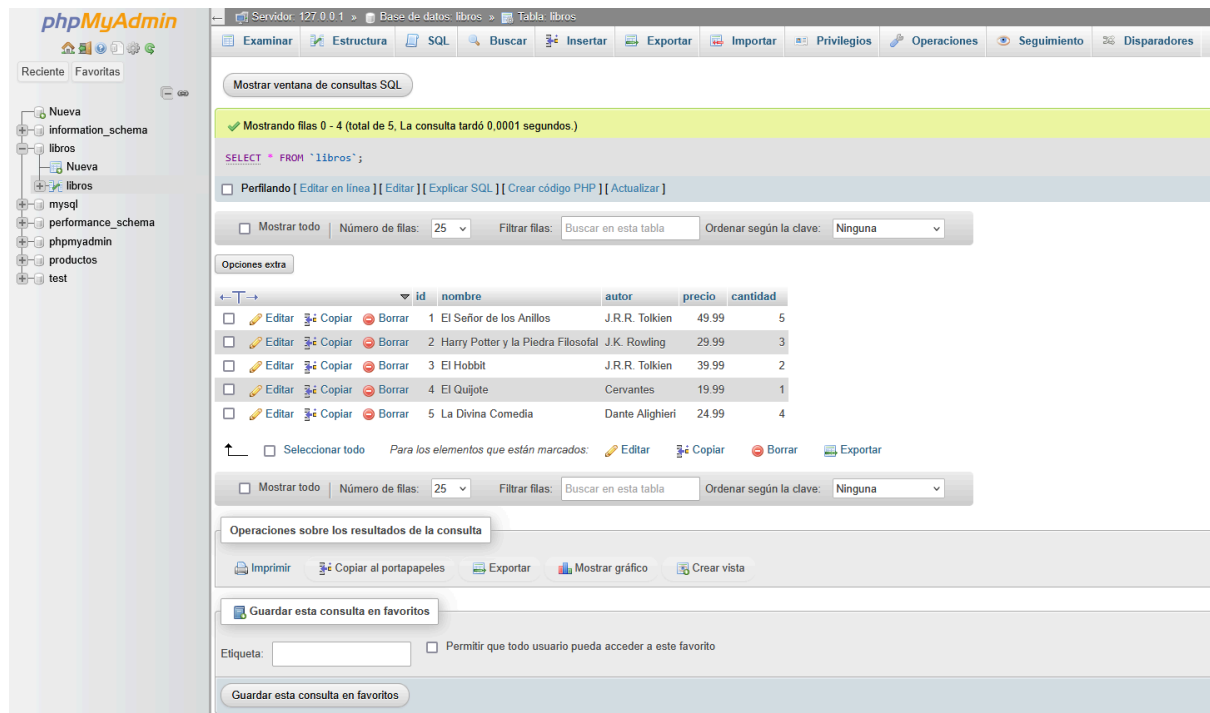
- **id**: Es la clave primaria que identifica de manera única cada libro.
- **nombre**: Campo obligatorio para el nombre del libro, con una longitud razonable.
- **autor**: Campo opcional que especifica el autor del libro.
- **precio**: Un valor decimal obligatorio para almacenar el precio con precisión.
- **cantidad**: Un número entero que no puede ser negativo, representa el inventario disponible.

## Ejemplo de Datos Iniciales:

Para realizar pruebas y demostraciones, se puede rellenar la base de datos con algunos registros:

```
11 INSERT INTO libros (id, nombre, autor, precio, cantidad) VALUES
12     (1, 'El Señor de los Anillos', 'J.R.R. Tolkien', 49.99, 5),
13     (2, 'Harry Potter y la Piedra Filosofal', 'J.K. Rowling', 29.99, 3),
14     (3, 'El Hobbit', 'J.R.R. Tolkien', 39.99, 2),
15     (4, 'El Quijote', 'Cervantes', 19.99, 1),
16     (5, 'La Divina Comedia', 'Dante Alighieri', 24.99, 4);
```

## Resultado mostrado en PHPMyAdmin lanzado por XAMPP:



## 2.2 Diagrama de Clases

El diseño del sistema incluirá clases que representen tanto los datos como las funcionalidades para gestionar los libros y conectar con la base de datos.

**Clases Identificadas:**

1. Clase **Libro**
2. Clase **BookshopSQLDAO** (SQL Data Access Object):
3. Clase **BookshopController**
4. Clase **BookshopApp**
5. Clase **BaseDatosException**
6. Clase **FormatoCampoException**

**Descripción de las clases:**

1. Clase **Libro**:
  - Contiene los atributos del libro: **id**, **titulo**, **autor**, **precio**, y **cantidad**.
  - Representa los datos directamente relacionados con la tabla **libro**.
2. Clase **BookshopSQLDAO** (SQL Data Access Object):
  - Proporciona métodos para abrir y cerrar conexiones.
  - Implementa las operaciones CRUD que interactúan directamente con la tabla **libro**.
3. Clase **BookshopController**:

- Actúa como intermediario entre la interfaz gráfica (ventana principal) y las clases de acceso a datos (**BookshopSQLDAO**).
- 4. **Clase BookshopApp:**
  - Representada por una ventana principal diseñada con **SceneBuilder**.
  - Permite a los usuarios interactuar con el sistema (ejemplo: botones para agregar, buscar, editar y eliminar libros).
- 5. **Clase BaseDatosException:**
  - Excepción lanzada cuando se produce un error al comunicarse con la base de datos.
- 6. **Clase FormatoCampoException:**
  - Excepción lanzada cuando en un campo se introducen datos en un formato no válido.

### 3. Herramientas y Tecnologías Utilizadas

#### 3.1 JavaFX

JavaFX es un framework de desarrollo de interfaces gráficas para aplicaciones de escritorio en Java. Ofrece un diseño moderno, soporta componentes avanzados como tablas y gráficos, y permite la integración de estilos mediante CSS para personalizar la apariencia de las interfaces. Su flexibilidad lo convierte en una excelente opción para crear GUIs dinámicas y responsivas.

#### 3.2 SceneBuilder

SceneBuilder es una herramienta visual que facilita el diseño de interfaces gráficas en JavaFX. Permite arrastrar y soltar componentes como botones, tablas y etiquetas, generando automáticamente el archivo FXML que define la estructura de la interfaz. Esto reduce la complejidad del diseño manual, acelerando el desarrollo de GUIs.

#### 3.3 MySQL

MySQL es un sistema de gestión de bases de datos relacional ampliamente utilizado. En este proyecto, almacena y organiza los datos relacionados con los libros. Su capacidad para manejar grandes volúmenes de datos y su compatibilidad con JDBC hacen de MySQL una opción robusta para la persistencia de información.

#### 3.4 JDBC (Java Database Connectivity)

JDBC es una API de Java que permite conectar aplicaciones con bases de datos relacionales como MySQL. Proporciona las herramientas necesarias para realizar operaciones como consultas SQL, inserciones, actualizaciones y eliminaciones. Es la tecnología que asegura la comunicación eficiente entre la aplicación JavaFX y la base de datos MySQL.

### 4. Desarrollo de la Interfaz Gráfica

#### 4.1 Estructura General de la GUI

La interfaz gráfica está organizada para ofrecer una experiencia intuitiva y funcional. La ventana principal incluye los siguientes elementos:

- **Zona superior:** Título de la aplicación y, opcionalmente, un menú para futuras funcionalidades.
- **Zona central:**
  - Tabla que muestra: ID, Título, Autor, Precio y Cantidad de cada Libro.
  - Campos de entrada (formularios) para agregar o actualizar libros.
- **Zona inferior:**
  - Botones para realizar las operaciones CRUD: *Agregar*, *Actualizar*, *Eliminar*.
  - Campo de búsqueda con un botón para filtrar libros según criterios.

## 4.2 Diseño de Pantallas y Componentes en SceneBuilder

El diseño de la interfaz fue realizado en **SceneBuilder** mediante componentes predefinidos, organizados de manera clara:

- **Pantalla Principal:**
  - **Tabla de Libros:** Componente TableView enlazado con la lista de libros.
  - **Formulario de Libro:** Campos de texto (TextField) para autor, título, precio y cantidad.
  - **Botones CRUD:**
    - Botón "Agregar" para añadir un nuevo libro.
    - Botón "Actualizar" para modificar un libro existente.
    - Botón "Eliminar" para borrar un libro seleccionado.
  - **Campo de Búsqueda:**
    - TextField para escribir criterios de búsqueda.
    - Botón "Buscar" para filtrar resultados en la tabla.

## 4.3 Navegación y Flujo de la Interfaz

El flujo de la interfaz está diseñado para ser sencillo y eficiente:

1. **Inicio:** Al abrir la aplicación, la tabla muestra todos los libros cargados desde la base de datos.
2. **Operaciones CRUD:**
  - El usuario selecciona un libro en la tabla para editar o eliminarlo.
  - Para agregar, completa los campos del formulario y presiona "Agregar".
3. **Filtrado:**
  - El usuario introduce un criterio en el campo de búsqueda y presiona "Buscar" para mostrar resultados específicos.
4. **Actualización Automática:**
  - Cada vez que se realiza una operación (CRUD o búsqueda), la tabla se actualiza automáticamente para reflejar los cambios.

## 5. Implementación de la Conexión a la Base de Datos

### 5.1 Configuración de la Base de Datos MySQL

## Creación de la Base de Datos:

1. Se define la base de datos **Libros** y la tabla **Libros** con los campos: **id**, **autor**, **titulo**, **precio**, y **cantidad**.
2. Se inicializa con datos de prueba para validar las operaciones CRUD.

## 5.2 Configuración de JDBC para la Conexión

### 1. Driver de Conexión:

- Descargar y añadir el **MySQL Connector/J** al proyecto (archivo **.jar**).
- Teniendo el proyecto abierto en Visual Studio Code, clicar abajo a la derecha en “Java Projects” > “+” > seleccionar los archivos de MySQL Connector/J
- El archivo settings.json ubicado en la carpeta .vscode del proyecto quedaría así:

```
1 "java.project.referencedLibraries": [  
2   "lib/**/*.jar",  
3   "resources\\mysql-connector-java-5.1.49\\mysql-connector-java-5.1.49.jar",  
4   "resources\\mysql-connector-java-5.1.49\\mysql-connector-java-5.1.49-bin.jar"  
5 ]
```

### 2. Driver de Conexión:

```
1 package dao;  
2  
3 import java.sql.Connection;  
4 import java.sql.DriverManager;  
5 import java.sql.PreparedStatement;  
6 import java.sql.ResultSet;  
7 import java.sql.SQLException;  
8 import java.sql.Statement;  
9 import java.util.Set;  
10 import java.util.TreeSet;  
11  
12 import model.Libro;  
13  
14 public class BookshopSQLDAO {  
15  
16     private final String IP_PORT = "localhost:3306";  
17     private final String DB_NAME = "libros";  
18     private final String URL = "jdbc:mysql://" + IP_PORT + "/" + DB_NAME + "?allowPublicKeyRetrieval=true&useSSL=false&serverTimezone=UTC";  
19     private final String USER = "pablo";  
20     private final String PASS = "1234";  
21  
22  
23 > public Set<Libro> getLibros() throws BaseDatosException { ...  
46  
47  
48 > /** ...  
57 > public void aniadirLibro(Libro libro) throws BaseDatosException { ...  
77  
78 > /** ...  
85 > public void modificarPrecioLibro(Libro libro, double nuevoPrecio) throws BaseDatosException { ...  
96  
97 > /** ...  
102 > public void eliminarLibro(Libro libro) throws BaseDatosException { ...  
116  
117 > /** ...  
128 > public Set<Libro> filtrarLibros(String[] filtros) throws FormatoCampoException, BaseDatosException { ...  
131 }
```

Para implementar este driver también se usará la extensión SQLTools de Visual Studio y establecer la conexión correctamente. Crearemos la conexión introduciendo los mismos datos que en el DAO:

SQLTools Settings

Connection Assistant

Connection Settings

Connection name\*

libros

Connection group

Connect using\*

Server and Port

Server Address\*

localhost

Port\*

3306

Database\*

libros

Username\*

pablo

SQLTOOLS

...

CONNECTIONS

+

SQL

↺

📄

MySQL

libros

pablo@localhost:3306/libros

🔌

Database\*

libros

Username\*

pablo

Password mode

Save as plaintext in settings

Password\*

....

MySQL driver specific options

Authentication Protocol

default

Try to switch protocols in case you have problems.

SSL

Disabled

Connection Timeout

Show records default limit

50

SAVE CONNECTION

## 5.3 Creación de Clases Modelo para la Interacción con la Base de Datos

1. Clase **Libro**:
  - Representa los datos de un libro:



```

1 package model;
2
3 public class Libro implements Comparable<Libro> {
4
5     private String nombre;
6     private String autor;
7     private int cantidad;
8     private double precio;
9     private int id;
10
11     public Libro(String nombre, String autor, int cantidad, double precio, int id) {
12         this.nombre = nombre;
13         this.autor = autor;
14         this.cantidad = cantidad;
15         this.precio = precio;
16         this.id = id;
17     }
18
19     public Libro(String nombre, String autor, int cantidad, double precio) {
20         this(nombre, autor, cantidad, precio, -1);
21     }
22
23     public String getNombre() { return nombre ;}
24     public String getAutor() { return autor ;}
25     public int getCantidad() { return cantidad ;}
26     public double getPrecio() { return precio ;}
27     public int getId() { return id ;}
28
29
30
31     public void setPrecio(double precio) {
32         this.precio = precio;
33     }
34
35     public void setId(int id) {
36         this.id = id;
37     }
38
39     @Override
40     public String toString() {
41         return autor + " " + nombre + " " + precio + " " + cantidad;
42     }
43
44     // metodos autogenerados por VSCode
45     @Override
46 > public int hashCode() { ...
54
55     @Override
56 > public boolean equals(Object obj) { ...
78
79 > public int compareTo(Libro l) { ...
82 }

```

## 2. Clase BookshopSQLDAO:

- Implementa las operaciones CRUD usando JDBC:

```

22  /**
23   * Obtiene todos los libros de la base de datos.
24   *
25   * @return Conjunto de libros obtenidos de la base de datos.
26   * @throws BaseDatosException si hubo un error al obtener los libros de la base de datos
27   */
28  public Set<Libro> getLibros() throws BaseDatosException {
29      Set<Libro> libros = new TreeSet<>();
30
31      try (Connection conn = DriverManager.getConnection(URL, USER, PASS);
32           Statement stmt = conn.createStatement()) {
33
34          String selectQuery = "select * from libros";
35          ResultSet rset = stmt.executeQuery(selectQuery);
36
37          while (rset.next()) {
38              int id = rset.getInt(columnLabel:"id");
39              String titulo = rset.getString(columnLabel:"titulo");
40              String autor = rset.getString(columnLabel:"autor");
41              double precio = rset.getDouble(columnLabel:"precio");
42              int cantidad = rset.getInt(columnLabel:"cantidad");
43
44              libros.add(new Libro(titulo, autor, cantidad, precio, id));
45          }
46      } catch (SQLException e) {
47          throw new BaseDatosException(message:"Error en la base de datos al obtener libros");
48      }
49      return libros;
50  }

```

```

53  /**
54   * Agrega un nuevo libro a la base de datos. El objeto Libro pasado como parametro
55   * contiene el titulo, autor, precio y cantidad del nuevo libro. La funcion no
56   * devuelve nada, simplemente se asegura que el nuevo libro se haya agregado a la
57   * base de datos correctamente.
58   *
59   * @param Libro nuevo libro a agregar a la base de datos
60   * @throws BaseDatosException si hubo un error al agregar el nuevo libro a la base de datos
61   */
62  public void aniadirLibro(Libro libro) throws BaseDatosException {
63      try ( Connection conn = DriverManager.getConnection(URL, USER, PASS);
64           Statement stmt = conn.createStatement(); ) {
65          // preparamos la sentencia de insercion para que reciba los valores de los
66          // campos del nuevo libro
67          String strInsert = "INSERT INTO libros(id, titulo, autor, precio, cantidad) VALUES (" +
68              libro.getId() + "," +
69              "'" + libro.getNombre() + "'," +
70              "'" + libro.getAutor() + "'," +
71              libro.getPrecio() + "," +
72              libro.getCantidad() + ")";
73          stmt.executeUpdate(strInsert);
74      } catch (SQLException e) {
75          throw new BaseDatosException(message:"Error en la base de datos al agregar libro");
76      }
77  }

```

```

80  /**
81   * Modifica el precio de un libro existente en la base de datos.
82   *
83   * @param Libro El libro que se va a modificar
84   * @param nuevoPrecio El nuevo precio del libro
85   * @throws BaseDatosException si hubo un error al modificar el libro en la base de datos
86   */
87  public void modificarPrecioLibro(Libro libro, double nuevoPrecio) throws BaseDatosException {
88      String updateStatement = "UPDATE libros SET precio = ? WHERE id = ?";
89      try (Connection conn = DriverManager.getConnection(URL, USER, PASS);
90           PreparedStatement pstmtUpdate = conn.prepareStatement(updateStatement, autoGeneratedKeys:1)) {
91          pstmtUpdate.setDouble(parameterIndex:1, nuevoPrecio);
92          pstmtUpdate.setInt(parameterIndex:2, libro.getId());
93          pstmtUpdate.executeUpdate();
94      } catch (SQLException e) {
95          throw new BaseDatosException(message:"Error en la base de datos al modificar el precio del libro");
96      }
97  }

```

```

100 /**
101  * Elimina un libro de la base de datos.
102  * @param Libro El libro a eliminar
103  * @throws BaseDatosException Si hubo un error en la base de datos
104  */
105  public void eliminarLibro(Libro libro) throws BaseDatosException {
106      String deleteStatement = "DELETE FROM libros WHERE titulo = ? AND autor = ? AND precio = ? AND cantidad = ?";
107      try (Connection conn = DriverManager.getConnection(URL, USER, PASS);
108           PreparedStatement pstmtDelete = conn.prepareStatement(deleteStatement, autoGeneratedKeys:1)) {
109
110          pstmtDelete.setString(parameterIndex:1, libro.getNombre());
111          pstmtDelete.setString(parameterIndex:2, libro.getAutor());
112          pstmtDelete.setDouble(parameterIndex:3, libro.getPrecio());
113          pstmtDelete.setInt(parameterIndex:4, libro.getCantidad());
114          pstmtDelete.executeUpdate();
115      } catch (SQLException e) {
116          throw new BaseDatosException(message:"Error en la base de datos al eliminar libro");
117      }
118  }

```

```

120 /**
121  * Filtra los libros en la base de datos segun los argumentos pasados
122  * @param filtros Una lista de cadenas que contienen los filtros a aplicar:
123  *      0: titulo
124  *      1: precio maximo
125  *      2: autor
126  *      3: cantidad minima
127  * @return Un conjunto de libros que cumplen con los filtros
128  * @throws FormatoCampoException si hubo un error en el formato de alguno de los filtros
129  * @throws BaseDatosException si hubo un error en la base de datos
130  */
131  public Set<Libro> filtrarLibros(String[] filtros) throws FormatoCampoException, BaseDatosException {
132      // TODO: pendiente implementar filtrado
133      return getLibros();
134  }
135
136  }

```

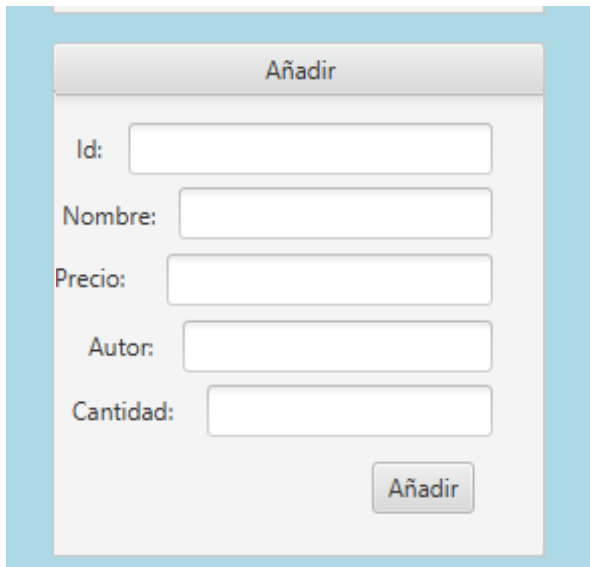
(pendiente de terminar)

## 6. Desarrollo de Funcionalidades CRUD

### 6.1 Inserción de Datos desde la Interfaz

La funcionalidad de inserción permite agregar nuevos libros desde un formulario en la GUI.

- **Flujo:** El usuario llena los campos correspondientes (título, autor, precio y cantidad) y presiona el botón "Añadir". El sistema valida los datos para asegurar que sean correctos (por ejemplo, que el precio sea positivo y que los campos no estén vacíos).



The image shows a screenshot of a web form titled "Añadir" (Add). The form is contained within a light blue border. It has five input fields, each with a label to its left: "Id:", "Nombre:", "Precio:", "Autor:", and "Cantidad:". Each label is in a light blue font. The input fields are white with a light gray border. At the bottom right of the form, there is a button labeled "Añadir" in a light gray box with a light blue border.

- **Resultado:** Si los datos son válidos, el libro se guarda en la base de datos y se actualiza automáticamente la tabla para mostrarlo.

## 6.2 Lectura de Datos y Visualización en la Tabla

La lectura permite mostrar todos los libros almacenados en la base de datos en un componente **ListView** de la GUI.

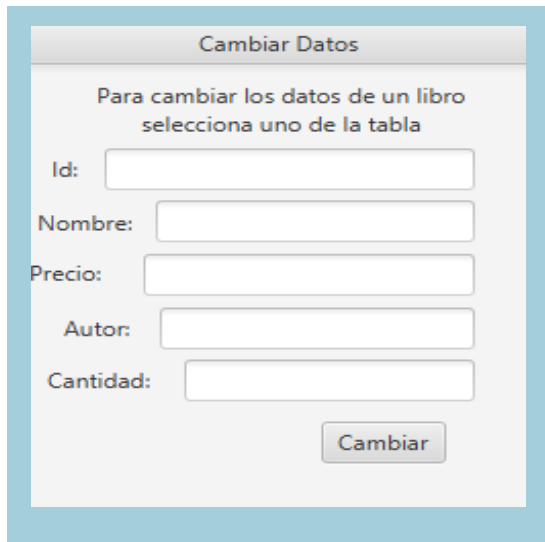
- **Flujo:** Al iniciar la aplicación o tras realizar operaciones como agregar, actualizar o eliminar libros, se ejecuta una consulta SQL que recupera los datos de la base de datos. Estos datos se organizan y se muestran en la lista.
- **Resultado:** Los libros se presentan con ID, nombre, descripción, precio y cantidad, permitiendo al usuario visualizar la información de forma clara.

(posible cambio por un TableView)

## 6.3 Actualización de Datos desde la GUI

La funcionalidad de actualización permite modificar la información de un libro existente.

- **Flujo:** El usuario selecciona un libro en la tabla. Los datos de este libro podrán ser cambiados. Una vez realizados los cambios, el usuario presiona el botón "Cambiar". El sistema valida los datos antes de guardarlos en la base de datos.

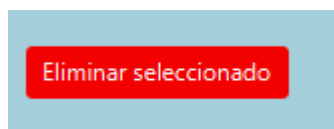


- **Resultado:** Si los datos son válidos, los cambios se reflejan tanto en la base de datos como en la tabla.

## 6.4 Eliminación de Datos y Confirmaciones de Borrado

La eliminación permite borrar un libro seleccionado tras la confirmación del usuario.

- **Flujo:** El usuario selecciona un libro en la tabla y presiona el botón "Eliminar". El sistema muestra un cuadro de confirmación para evitar eliminaciones accidentales. Si el usuario confirma, el libro se elimina de la base de datos.

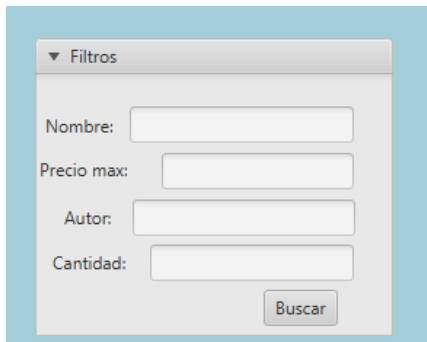


- **Resultado:** El libro desaparece tanto de la base de datos como de la tabla. La interfaz se actualiza automáticamente para reflejar los cambios.

## 7. Implementación de Filtrado y Búsqueda de Datos

### 7.1 Diseño de Formularios de Filtrado

Se crean formularios en la GUI que permiten a los usuarios ingresar criterios de búsqueda o filtros. Estos formularios incluyen campos como el nombre del libro, rangos de precio o cantidad mínima, proporcionando flexibilidad para personalizar las búsquedas.



▼ Filtros

Nombre:

Precio max:

Autor:

Cantidad:

Buscar

## 7.2 Generación de Consultas SQL Dinámicas para Búsquedas

El sistema construye consultas SQL dinámicas en función de los criterios ingresados por el usuario. Si un campo de filtro está vacío, el sistema lo omite de la consulta, permitiendo búsquedas amplias o específicas según sea necesario.

## 7.3 Visualización de Resultados Filtrados

Los resultados de la búsqueda se muestran en el componente **ListView** de la interfaz gráfica. Esto permite al usuario visualizar solo los libros que cumplen con los criterios seleccionados, proporcionando una herramienta eficiente para la gestión de datos.

## 8. Bibliografía

Chua Hock-Chuan (2024). Java Database Programming (JDBC) by Examples with MySQL  
[https://www3.ntu.edu.sg/home/ehchua/programming/java/JDBC\\_Basic.html](https://www3.ntu.edu.sg/home/ehchua/programming/java/JDBC_Basic.html)

Eugen Paraschiv, Kevin Gilmore (2024). The DAO Pattern in Java  
<https://www.baeldung.com/java-dao-pattern>

Getting Started with JavaFX <https://openjfx.io/openjfx-docs/#IDE-VSCode>

Chua Hock-Chuan (2021). Programming Graphical User Interface (GUI) Part 1  
[https://www3.ntu.edu.sg/home/ehchua/programming/java/J4a\\_GUI.html](https://www3.ntu.edu.sg/home/ehchua/programming/java/J4a_GUI.html)

Chua Hock-Chuan (2021). Programming Graphical User Interface (GUI) Part 2  
[https://www3.ntu.edu.sg/home/ehchua/programming/java/J4a\\_GUI\\_2.html](https://www3.ntu.edu.sg/home/ehchua/programming/java/J4a_GUI_2.html)

Chua Hock-Chuan (2021). Programming Graphical User Interface (GUI) Part 3  
[https://www3.ntu.edu.sg/home/ehchua/programming/java/J4a\\_GUI\\_3.html](https://www3.ntu.edu.sg/home/ehchua/programming/java/J4a_GUI_3.html)