

1ºDAM Programación

# MUNDOFORO

## 3ºEV Proyecto Final

---



---

<b>Índice.....</b>	<b>2</b>
1. Resumen del Proyecto.....	3
2. Fase de Diseño.....	4
Modelo Entidad-Relación (ER).....	4
Casos de Uso.....	4
Diagrama de Clases.....	5
Diagrama de Pantallas.....	6
3. Fase de Desarrollo.....	12
Temporalización.....	12
Código Destacado.....	12
Tecnologías y Herramientas usadas.....	13
Mejoras.....	13
Repositorio de GitHub.....	13
4. Lista de Comprobación.....	13
Herencia.....	13
Relaciones 1:N.....	14
DAO.....	15
Mostrar.....	16
Insertar.....	18
Eliminar.....	20
Actualizar.....	21

---

## 1. Resumen del Proyecto

Este proyecto consiste en una aplicación desarrollada en Java, utilizando JavaFX para la interfaz gráfica. Simula un foro con funcionalidades de autenticación y registro de usuarios.

Funcionalidad según el rol del usuario

- Usuario creador:
  - Puede crear, actualizar y eliminar foros, pero únicamente aquellos que él mismo haya creado.
  - La aplicación mantiene un registro del número total de foros creados por cada usuario.
  - Tiene opción de eliminar su propio usuario
- Usuario común:
  - Puede comentar en los foros existentes.
  - Tiene la opción de eliminar su propio usuario.
  - Tiene un nivel de participación, el cual se determina según la cantidad de comentarios publicados.

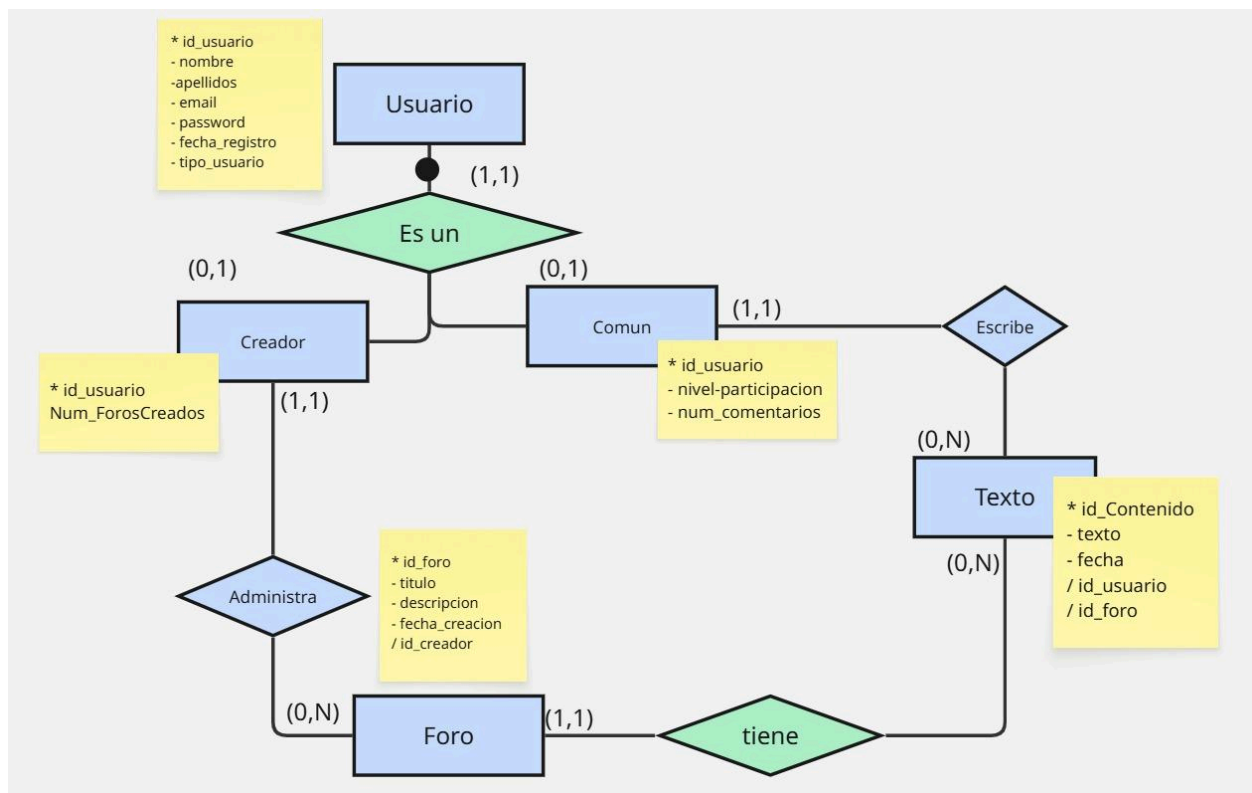
Interfaz Gráfica

- En el panel izquierdo, se muestra un listado de foros:
  - Los usuarios comunes pueden ver todos los foros disponibles.
  - Los usuarios creadores solo visualizan los foros que han creado.
- En el panel derecho, se visualizan los comentarios del foro seleccionado:
  - Los usuarios comunes pueden ver y añadir comentarios.
  - Los usuarios creadores pueden ver los comentarios, pero no pueden añadir nuevos.

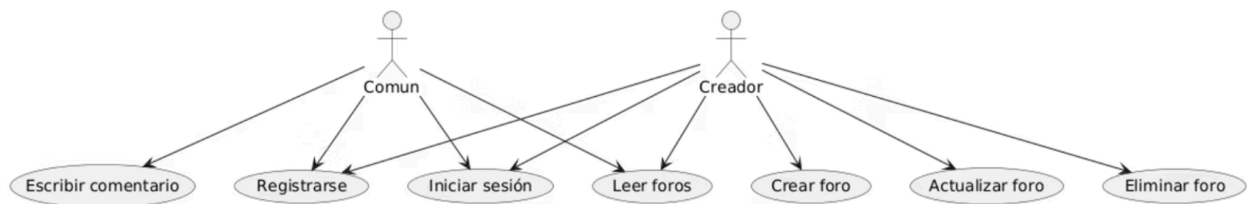
- Además, en la interfaz del usuario creador se muestran botones para añadir, actualizar y eliminar foros:
  - El botón de actualizar despliega una lista (menú desplegable) que solo incluye los foros creados por ese usuario.

## 2. Fase de Diseño

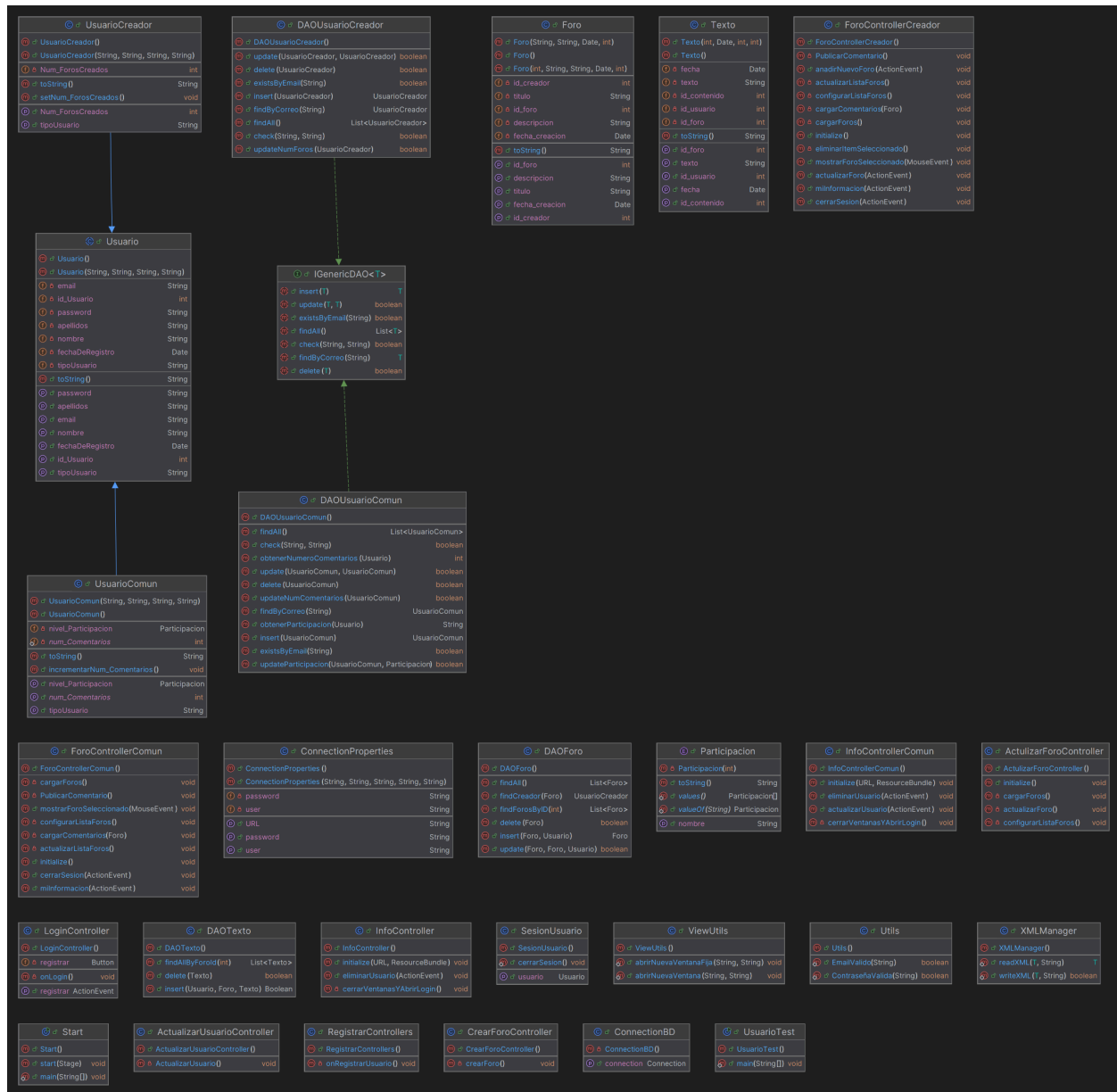
### Modelo Entidad-Relación (ER)



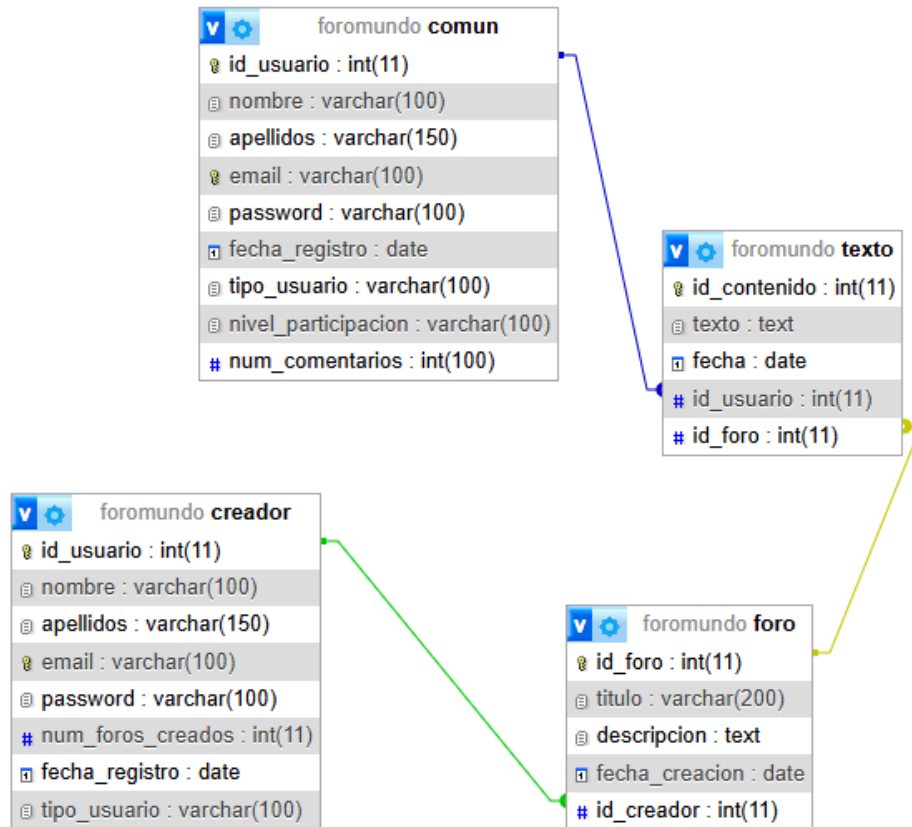
### Casos de Uso



## Diagrama de Clases

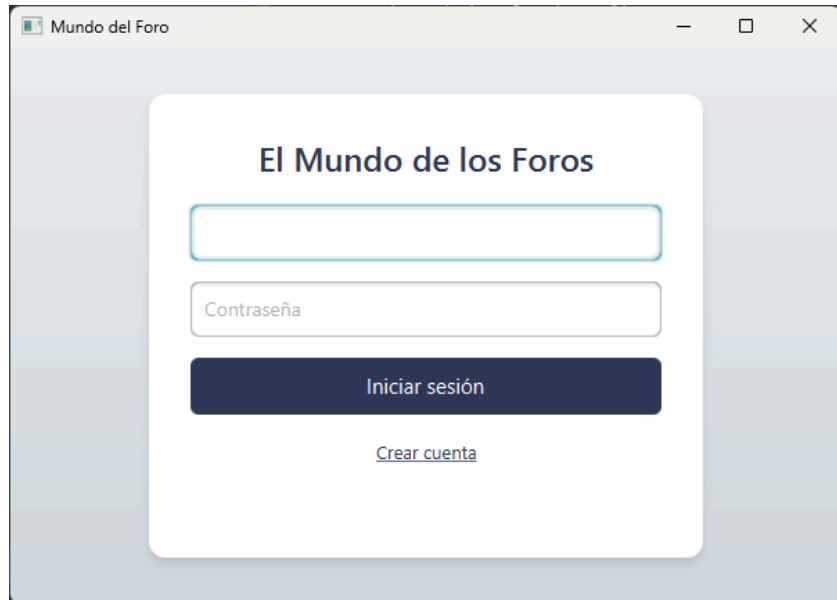
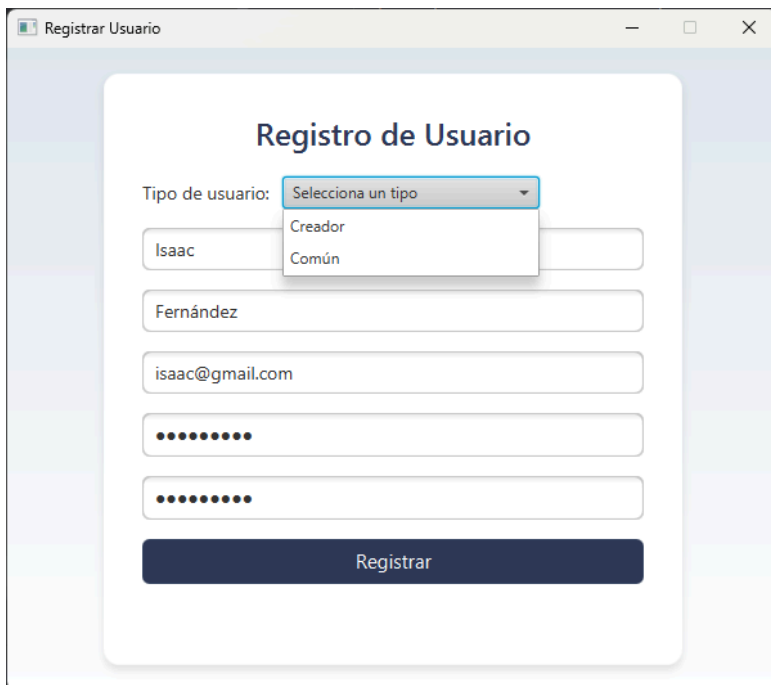


## Estructura SQL



## Diagrama de Pantallas

Indicaremos el programa. Primero nos aparece la pestaña de inicio de sesión, en la cual tenemos los dos campos para meter nuestras credenciales (Email y Contraseña). En nuestro caso, vamos a crear un nuevo usuario pulsando en el enlace de "Crear Cuenta".

A screenshot of a web browser window titled "Mundo del Foro". The main content is a white card with the title "El Mundo de los Foros" in bold blue text. Below the title are two input fields: the first is empty, and the second is labeled "Contraseña". Below these fields is a dark blue button with the text "Iniciar sesión". At the bottom of the card is a blue link that says "Crear cuenta".A screenshot of a web browser window titled "Registrar Usuario". The main content is a white card with the title "Registro de Usuario" in bold blue text. Below the title is a label "Tipo de usuario:" followed by a dropdown menu showing "Selecciona un tipo" with options "Creador" and "Común". Below this are four input fields: the first contains "Isaac", the second contains "Fernández", the third contains "isaac@gmail.com", and the fourth is a password field with dots. At the bottom of the card is a dark blue button with the text "Registrar".

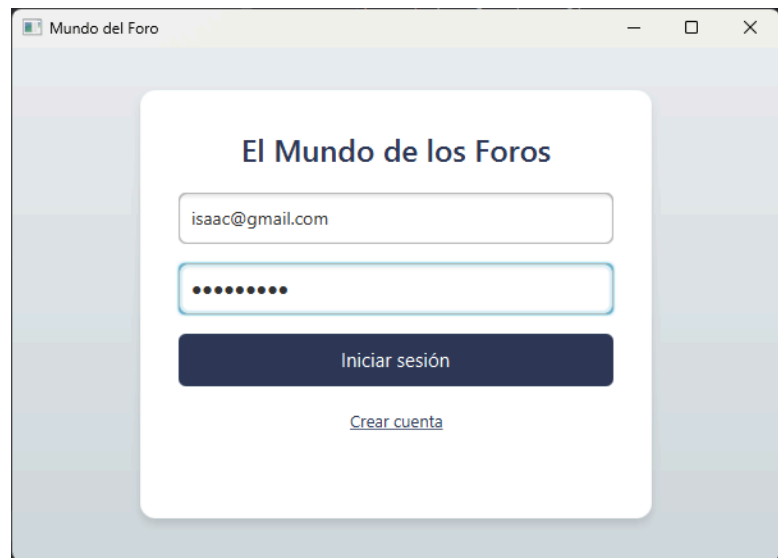
Se nos abrirá una nueva ventana donde podremos, en un desplegable, seleccionar el tipo de usuario que queremos crear: Creador o Común. En mi caso, vamos a crear un Común. A continuación, rellenamos todos los campos.

Es importante destacar que el correo es un atributo único, es decir, no puede haber dos cuentas con el mismo correo. Si por casualidad pasa eso, nos saldrá un mensaje de error.

Además, la contraseña tiene restricciones de sintaxis para ser segura, y si no se cumplen, también aparecerá un mensaje de error. Si las contraseñas no coinciden, lo mismo: salta error.

Una vez creado el usuario, iniciamos sesión.

En nuestro caso, vamos a iniciar con el usuario creador registrado anteriormente.

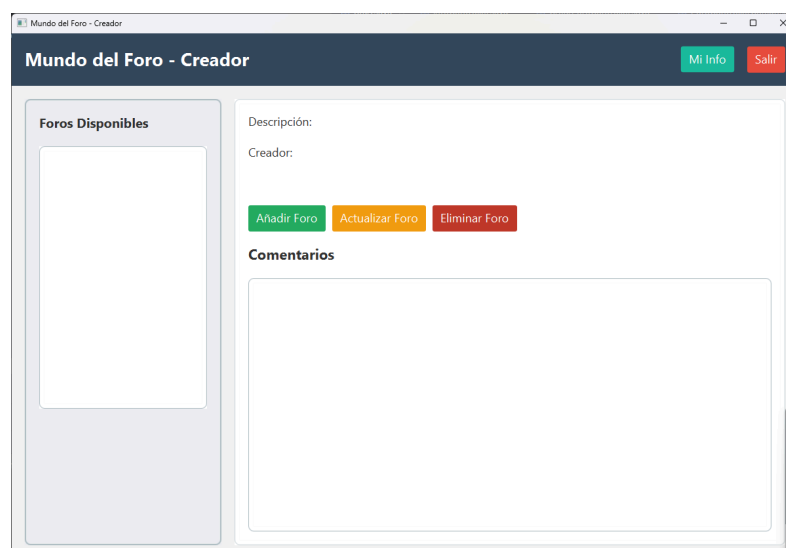


Se nos abrirá la pestaña en modo creador. En ella podemos ver, en el lateral izquierdo, una lista de foros. En este caso, solo aparecen nuestros propios foros.

En el lado derecho, vemos la descripción y el creador del foro, que en nuestro caso será siempre el mismo.

Abajo de los detalles aparecen tres botones: Añadir, Actualizar y Eliminar. Y más abajo, los comentarios del foro seleccionado.

Además, en la parte superior derecha hay dos botones: uno de "Mi información" y otro de "Cerrar sesión".



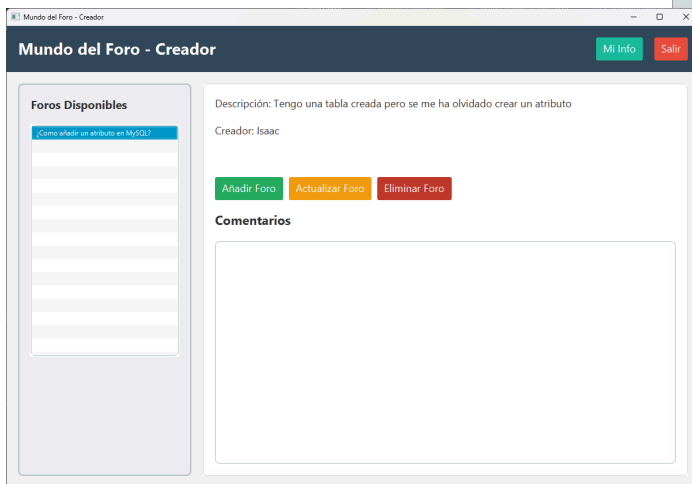
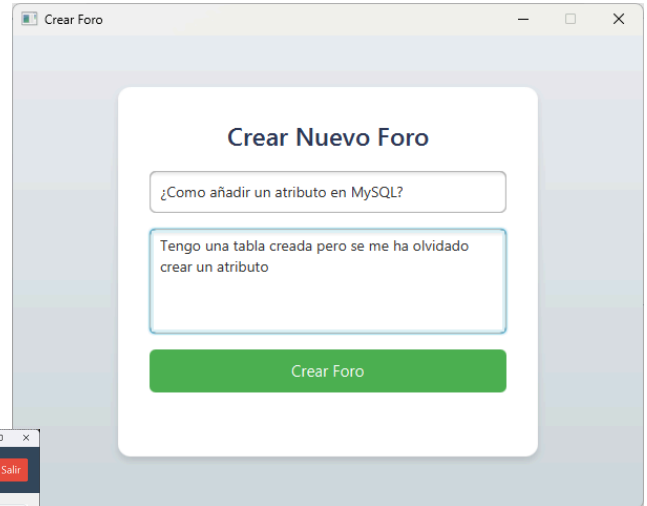


Comenzamos creando un foro pulsando en el botón verde.

Se nos abrirá una ventana con dos apartados: uno para el título y otro para la descripción.

Si no rellenamos los dos campos, aparecerá un error.

Una vez rellenados, pulsamos en "Crear Foro", se cerrará la ventana y la lista de la izquierda se actualiza automáticamente.




¿Nos hemos equivocado o queremos modificar algo del foro?

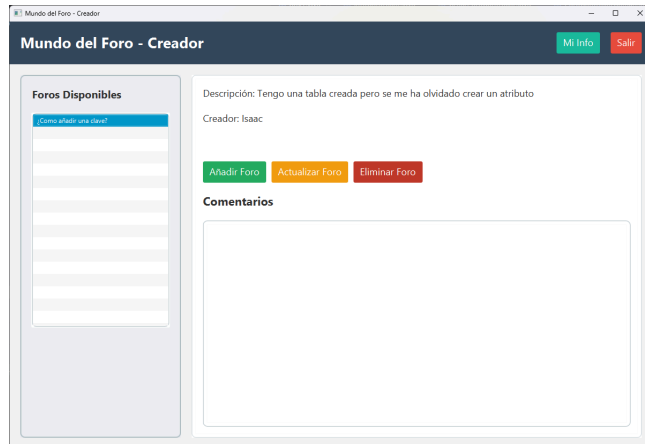
Entonces pulsamos el botón amarillo de "Actualizar".

Se nos abrirá una nueva ventana parecida a la de crear foro, lo único distinto es que tiene un desplegable donde aparecen solo nuestros foros. Seleccionamos el que queremos actualizar.

En esta ventana no es obligatorio actualizar ambos campos.

Una vez le damos al botón de actualizar, se actualiza la lista.



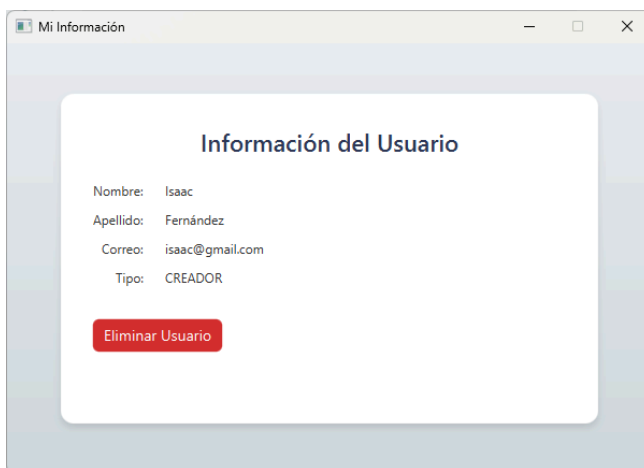
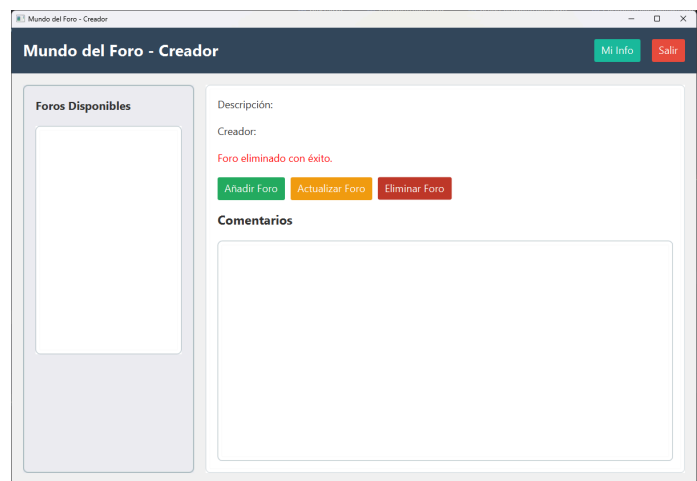


Ahora vamos a eliminar el foro.

Para eso, seleccionamos el foro y pulsamos el botón rojo de eliminar.

Hay que destacar que también se puede eliminar un comentario, simplemente seleccionándolo y pulsando en eliminar.

Ahora vamos a ver nuestra información, usando el botón "Mi info".



Se nos abre una nueva ventana donde aparecen los datos principales del usuario.

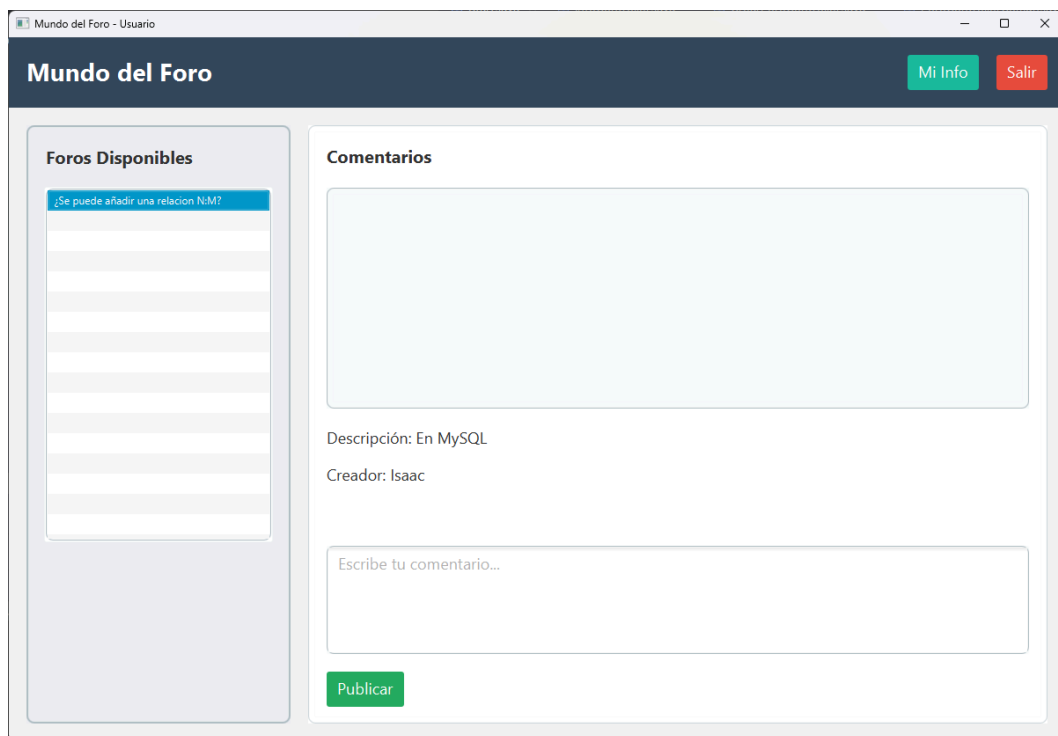
También tenemos un botón para eliminar el usuario; al pulsarlo, ese usuario será eliminado.

Y para finalizar con el usuario creador, cerramos sesión usando el botón de salir de la pantalla principal

Ahora iniciaremos sesión con el usuario común que he creado anteriormente.



Se nos abrirá una pestaña diferente a la del creador. En este caso, nos aparece el listado de todos los foros, la lista de todos los comentarios del foro que tengamos seleccionado, y un pequeño campo para comentar, además de los botones de "Mi info" y "Salir", que también vimos en el usuario creador.

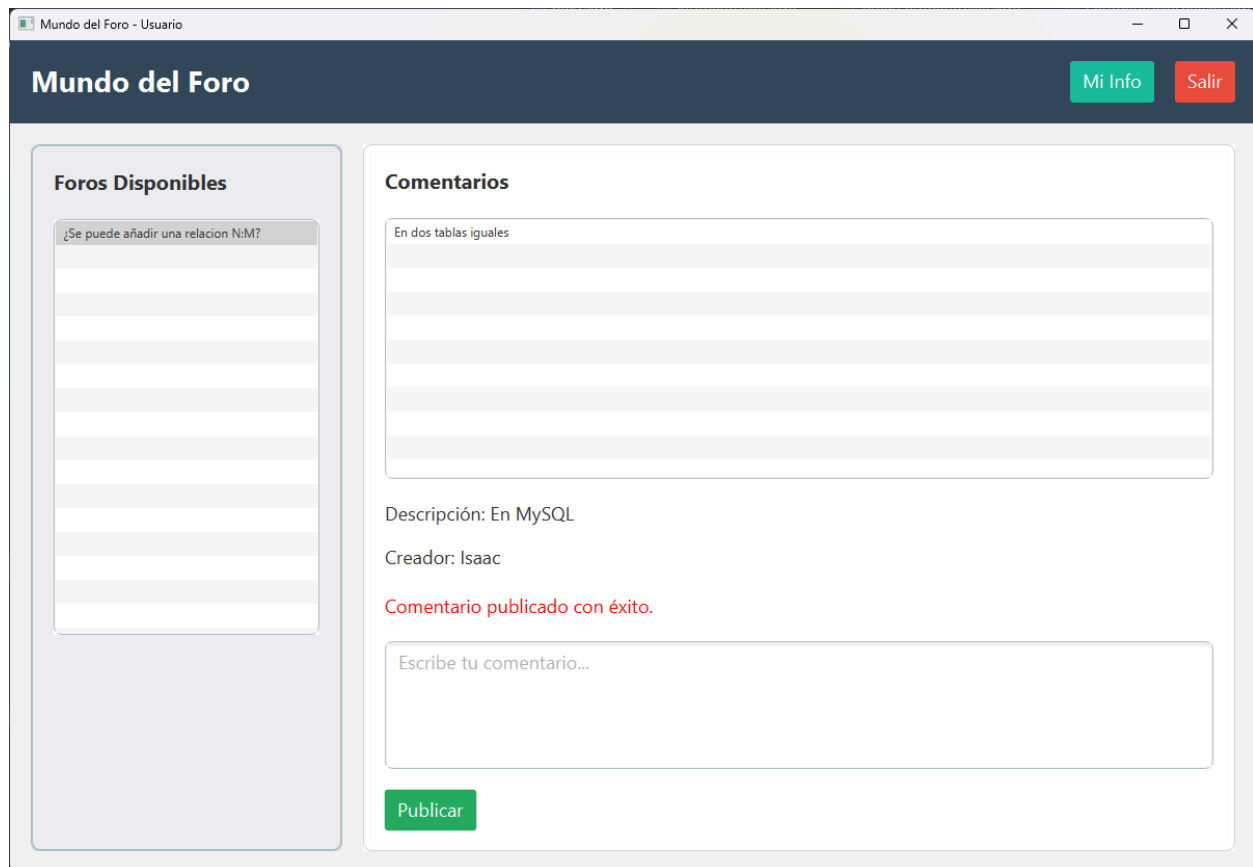


---

Vamos a escribir un comentario.

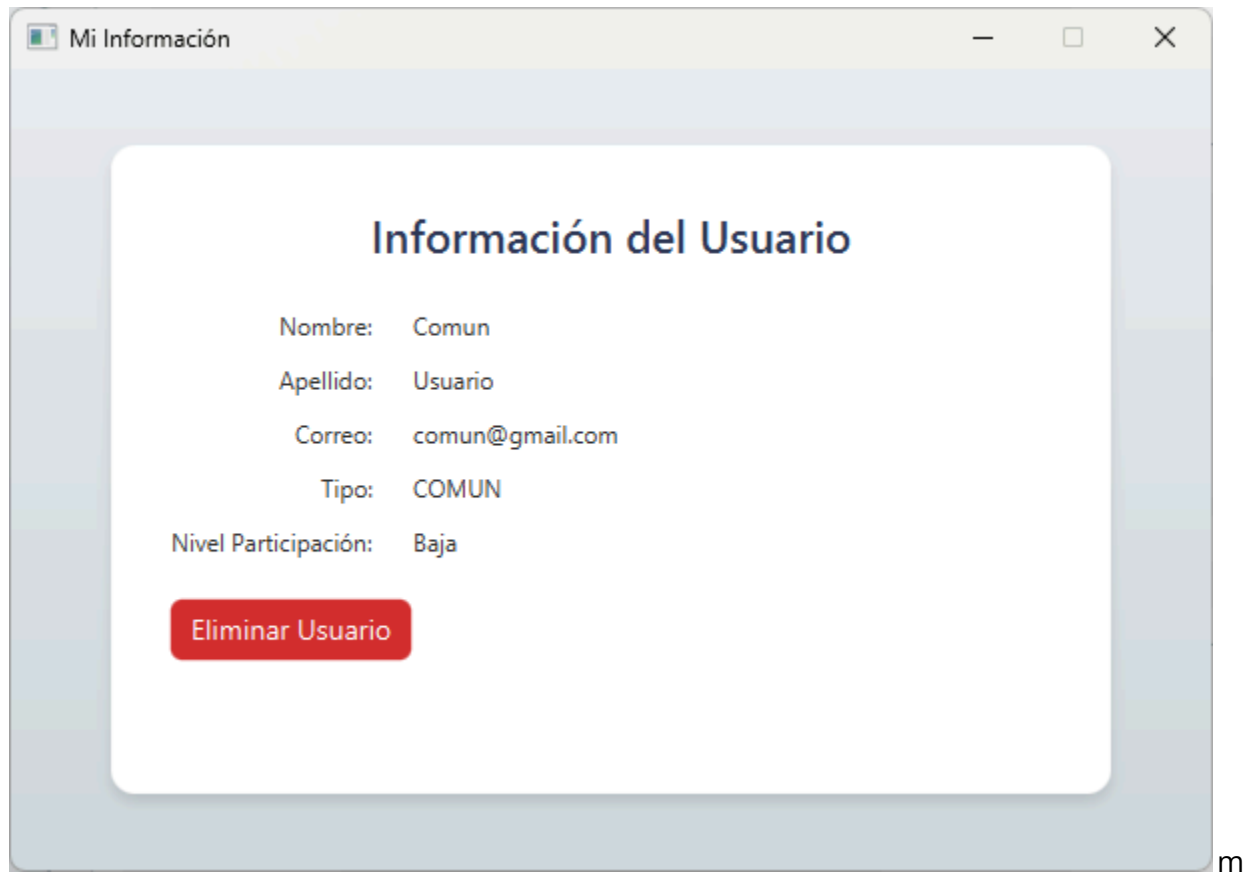
Escribimos en el campo de comentario y pulsamos en "Publicar".

Si hay algún error, como por ejemplo que no hay foro seleccionado, no hay comentario escrito o no se ha podido publicar, nos aparecerá un mensaje de error.



Ahora vamos a observar la ventana de "Mi info", ya que es diferente a la del creador.

Pulsando en "Mi info" nos aparece la ventana con nuestra información, donde aparecen los mismos campos que antes, pero además incluye un campo extra llamado "Nivel de participación", que va cambiando dependiendo de los comentarios publicados.



### 3. Fase de Desarrollo

#### Temporalización

- **Sprint 1 : Conexión a BD**
  - Modelo ER, diagrama clases, conexión GitHub
- **Sprint 2: Desarrollo de clases**
  - Clases modelo, DAOs, base datos
- **Sprint 3: Desarrollo de Interfaz Gráfica**
  - Controladores , Vista (FXML)
- **Sprint 4: Documentar**
  - Depurado y comentado código , finalización de Documentación

---

### Código Destacado

- Validación de Email y Contraseña a través de expresiones Lógico
- Definir Listado de Foro según el usuario iniciado
- Niveles de Participación de Usuarios Comunes
- Creación de Interfaz Genérica para los DAOS Usuarios

### Tecnologías y Herramientas usadas

Java 11+, JavaFX, FXML, Maven, MySQL, JAXB, GitHub, Discord, ChatGPT

### Mejoras

- Opción de unirse a un Foro , para poder comentar
- Dividir Foros en categorías y acceder atreves de categorías
- Realizar SubForos de un mismo tema comunicativo
- Opción de añadir multimedia

### Repositorio de GitHub

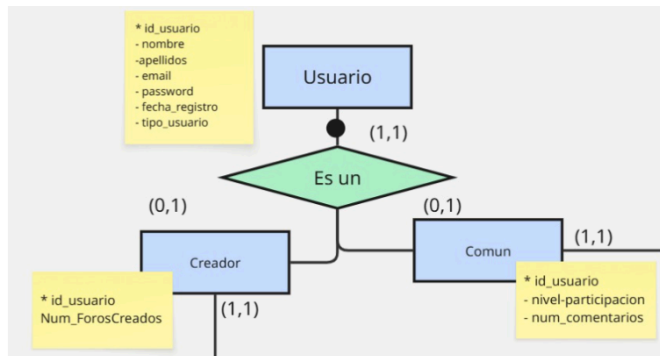
El repositorio almacena el código atreves de versiones y ramas de nuestro proyecto

[\*\*EV3-ElMundoDelForo\*\*](#)

## 4. Lista de Comprobación

### Herencia

La herencia está utilizada en el Usuario. En él tenemos dos tipos de usuario: Común y Creador, y cada uno tiene atributos específicos (Común: num\_comentarios, nivel\_participacion; Creador: num\_foros\_creados). Luego está la clase abstracta Usuario, que contiene los atributos y métodos comunes para ambos tipos de usuarios.



```
UsuarioComun.java
package es.isfranciscolesosrios.dami.isaac.ev3elmundodelforo.modelo;

// La clase UsuarioComun extiende la clase Usuario.
// Representa un usuario de tipo "COMUN" que tiene un número de comentarios y un nivel de participación.
public class UsuarioComun extends Usuario {
    @Override
    private static int num_comentarios = 0; // Usages
    private Participacion nivel_participacion; // Usages

    /**
     * Constructor vacío de la clase UsuarioComun.
     * Llama al constructor de la clase base Usuario.
     */
    public UsuarioComun() {
        super();
    }

    /**
     * Constructor de la clase UsuarioComun con parámetros.
     *
     * @param nombre El nombre del usuario.
     * @param apellidos Los apellidos del usuario.
     * @param email El correo electrónico del usuario.
     * @param password La contraseña del usuario.
     */
    public UsuarioComun(String nombre, String apellidos, String email, String password) {
        super(nombre, apellidos, email, password);
    }

    /**
     * Obtiene el número de comentarios realizados por el usuario.
     *
     * @return El número de comentarios realizados por el usuario.
     */
    public int getNum_comentarios() {
        return num_comentarios;
    }

    /**
     * Obtiene el nivel de participación del usuario.
     *
     * @return El nivel de participación del usuario como un valor de tipo Participacion.
     */
    public Participacion getNivel_Participacion() {
        return nivel_participacion;
    }

    /**
     * Incrementa el número de comentarios realizados por el usuario.
     *
     * @return El nuevo número de comentarios después de la actualización.
     */
    public void setNum_comentarios(int num_comentarios) {
        this.num_comentarios = num_comentarios;
    }

    /**
     * Incrementa el número de comentarios.
     */
    public void incrementarNum_comentarios() {
        this.num_comentarios++;
    }

    /**
     * Devuelve el tipo de usuario como un String.
     *
     * @return El tipo de usuario, en este caso siempre devuelve "COMUN".
     */
    @Override
    public String getTipo_usuario() {
        return "COMUN";
    }

    /**
     * Devuelve una representación en cadena del objeto UsuarioComun.
     *
     * @return Una cadena que representa al usuario, mostrando su nombre.
     */
    @Override
    public String toString() {
        return "UsuarioComun{" +
            "Nombre=" + getNombre() +
            "Apellidos=" + getApellidos() +
            "Email=" + getEmail() +
            "Password=" + getPassword() +
            "Fecha_registro=" + getFecha_registro() +
            "Tipo_usuario=" + getTipo_usuario() +
            "Num_comentarios=" + getNum_comentarios() +
            "Nivel_participacion=" + getNivel_Participacion() +
            "}";
    }
}
```

```
UsuarioCreador.java
package es.isfranciscolesosrios.dami.isaac.ev3elmundodelforo.modelo;

import java.sql.Date;

// La clase UsuarioCreador extiende la clase Usuario.
// Representa un usuario de tipo "CREADOR" que tiene un número de foros creados.
public class UsuarioCreador extends Usuario {
    @Override
    private int num_foros_creados; // Usages

    /**
     * Constructor vacío de la clase UsuarioCreador.
     * Llama al constructor de la clase base Usuario.
     */
    public UsuarioCreador() {
        super();
    }

    /**
     * Constructor de la clase UsuarioCreador con parámetros.
     *
     * @param nombre El nombre del usuario.
     * @param apellidos Los apellidos del usuario.
     * @param email El correo electrónico del usuario.
     * @param password La contraseña del usuario.
     */
    public UsuarioCreador(String nombre, String apellidos, String email, String password) {
        super(nombre, apellidos, email, password);
    }

    /**
     * Obtiene el número de foros creados por el usuario.
     *
     * @return El número de foros creados por el usuario.
     */
    public int getNum_foros_creados() {
        return num_foros_creados;
    }

    /**
     * Incrementa el número de foros creados por el usuario.
     *
     * @return El nuevo número de foros creados después de la actualización.
     */
    public void setNum_foros_creados(int num_foros_creados) {
        this.num_foros_creados = num_foros_creados;
    }

    /**
     * Devuelve el tipo de usuario como un String.
     *
     * @return El tipo de usuario, en este caso siempre devuelve "CREADOR".
     */
    @Override
    public String getTipo_usuario() {
        return "CREADOR";
    }

    /**
     * Devuelve una representación en cadena del objeto UsuarioCreador.
     *
     * @return Una cadena con el número de foros creados por el usuario.
     */
    @Override
    public String toString() {
        return "UsuarioCreador{" +
            "Nombre=" + getNombre() +
            "Apellidos=" + getApellidos() +
            "Email=" + getEmail() +
            "Password=" + getPassword() +
            "Fecha_registro=" + getFecha_registro() +
            "Tipo_usuario=" + getTipo_usuario() +
            "Num_foros_creados=" + getNum_foros_creados() +
            "}";
    }
}
```

```
Usuario.java
package es.isfranciscolesosrios.dami.isaac.ev3elmundodelforo.modelo;

import java.sql.Date;
import java.util.Date;

// La clase abstracta Usuario representa la estructura básica de un usuario en la aplicación.
// Contiene atributos comunes como nombre, apellidos, email, contraseña, y la fecha de registro.
// Esta clase será extendida por otras clases como UsuarioComun y UsuarioCreador.
public abstract class Usuario {
    @Override
    private int id_usuario; // Usages
    private String nombre; // Usages
    private String apellidos; // Usages
    private String email; // Usages
    private String password; // Usages
    private Date fecha_registro; // Usages
    private String tipo_usuario; // Usages

    /**
     * Constructor de la clase Usuario que recibe nombre, apellidos, email y contraseña.
     *
     * @param nombre Nombre del usuario.
     * @param apellidos Apellidos del usuario.
     * @param email Correo electrónico del usuario.
     * @param password Contraseña del usuario.
     */
    public Usuario(String nombre, String apellidos, String email, String password) {
        this.nombre = nombre;
        this.apellidos = apellidos;
        this.email = email;
        this.password = password;
    }

    /**
     * Constructor vacío de la clase Usuario. Se usa para inicializar un objeto de tipo Usuario
     * sin proporcionar valores de los atributos.
     */
    public Usuario() {
        //
    }

    /**
     * Obtiene el ID del usuario.
     *
     * @return El ID del usuario.
     */
    public int getId_usuario() {
        return id_usuario;
    }

    /**
     * Establece el ID del usuario.
     *
     * @param id_usuario El ID del usuario a establecer.
     */
    public void setId_usuario(int id_usuario) {
        this.id_usuario = id_usuario;
    }

    /**
     * Obtiene el nombre del usuario.
     *
     * @return El nombre del usuario.
     */
    public String getNombre() {
        return nombre;
    }

    /**
     * Establece el nombre del usuario.
     *
     * @param nombre El nombre del usuario a establecer.
     */
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    /**
     * Obtiene los apellidos del usuario.
     *
     * @return Los apellidos del usuario.
     */
    public String getApellidos() {
        return apellidos;
    }

    /**
     * Establece los apellidos del usuario.
     *
     * @param apellidos Los apellidos del usuario a establecer.
     */
    public void setApellidos(String apellidos) {
        this.apellidos = apellidos;
    }

    /**
     * Obtiene el correo electrónico del usuario.
     *
     * @return El correo electrónico del usuario.
     */
    public String getEmail() {
        return email;
    }

    /**
     * Establece el correo electrónico del usuario.
     *
     * @param email El correo electrónico del usuario a establecer.
     */
    public void setEmail(String email) {
        this.email = email;
    }

    /**
     * Obtiene la contraseña del usuario.
     *
     * @return La contraseña del usuario.
     */
    public String getPassword() {
        return password;
    }

    /**
     * Establece la contraseña del usuario.
     *
     * @param password La contraseña del usuario a establecer.
     */
    public void setPassword(String password) {
        this.password = password;
    }

    /**
     * Obtiene la fecha de registro del usuario.
     *
     * @return La fecha de registro del usuario.
     */
    public Date getFecha_registro() {
        return fecha_registro;
    }

    /**
     * Establece la fecha de registro del usuario.
     *
     * @param fecha_registro La fecha de registro del usuario a establecer.
     */
    public void setFecha_registro(Date fecha_registro) {
        this.fecha_registro = fecha_registro;
    }

    /**
     * Obtiene el tipo de usuario (por ejemplo, "COMUN", "CREADOR").
     *
     * @return El tipo de usuario.
     */
    public String getTipo_usuario() {
        return tipo_usuario;
    }

    /**
     * Establece el tipo de usuario.
     *
     * @param tipo_usuario El tipo de usuario a establecer.
     */
    public void setTipo_usuario(String tipo_usuario) {
        this.tipo_usuario = tipo_usuario;
    }

    /**
     * Devuelve una representación en cadena de este objeto Usuario.
     *
     * @return Una cadena con los atributos del usuario, como su ID, nombre, apellidos, email y fecha de registro.
     */
    @Override
    public String toString() {
        return "Usuario{" +
            "id_usuario=" + getId_usuario() +
            "nombre=" + getNombre() +
            "apellidos=" + getApellidos() +
            "email=" + getEmail() +
            "password=" + getPassword() +
            "fecha_registro=" + getFecha_registro() +
            "tipo_usuario=" + getTipo_usuario() +
            "}";
    }
}
```

## Relaciones 1:N

La relación 1:N que tenemos es entre UsuarioCreador y Foro, ya que un creador puede crear 0 o varios foros.

También tenemos una relación entre Texto y Foro, porque un foro puede tener 0 o varios textos.

Y por último, hay una relación entre Texto y UsuarioComun, ya que un usuario común puede escribir 0 o N textos.

## DAO

Los DAOs que he creado en mi proyecto son para controlar: Foro, Texto, UsuarioComún, UsuarioCreador y una interfaz genérica llamada IGenericDAO.

Os voy a explicar cómo funciona la interfaz genérica IGenericDAO usando como ejemplo el DAOComun.

Pues bien, IGenericDAO es una interfaz genérica que incluye métodos como: update, delete, findByCorreo, findAll, check, existsByEmail. Esta interfaz está pensada específicamente para ser usada tanto por DAOComun como por DAOCreador.

En DAOComun, además de las consultas obligatorias que vienen de la interfaz, también hay métodos específicos para manejar los datos del usuario común, como los valores de comentarios y nivel de participación.

```
package es.iesfrancombaforos.dao.impl;

import java.util.List;

public class DAOComun implements IGenericDAO {

    private final String URL_ORIGEN_USUARIO_COMUN = "SELECT * FROM USUARIO_COMUN WHERE ID_USUARIO = ?";
    private final String URL_ORIGEN_USUARIO_CREADOR = "SELECT * FROM USUARIO_CREADOR WHERE ID_USUARIO = ?";
    private final String URL_ORIGEN_FORO = "SELECT * FROM FORO WHERE ID_FORO = ?";
    private final String URL_ORIGEN_TEXTO = "SELECT * FROM TEXTO WHERE ID_TEXTO = ?";
    private final String URL_ORIGEN_PARTICIPACION = "SELECT * FROM PARTICIPACION WHERE ID_PARTICIPACION = ?";
    private final String URL_ORIGEN_COMENTARIO = "SELECT * FROM COMENTARIO WHERE ID_COMENTARIO = ?";

    // Inserta un nuevo usuario de tipo UsuarioComun en la base de datos.
    // Devuelve el ID del usuario insertado con los datos del nuevo usuario.
    // Devuelve un mensaje de error si la operación falla.
    // Devuelve SQLException si ocurre un error con la base de datos.
    public int insertarUsuarioComun(UsuarioComun usuario) throws SQLException {
        // Inserta un nuevo usuario de tipo UsuarioComun en la base de datos.
        // Devuelve el ID del usuario insertado con los datos del nuevo usuario.
        // Devuelve un mensaje de error si la operación falla.
        // Devuelve SQLException si ocurre un error con la base de datos.
    }

    // Actualiza los datos de un usuario de tipo UsuarioComun en la base de datos.
    // Devuelve un mensaje de error si la operación falla.
    // Devuelve SQLException si ocurre un error con la base de datos.
    public boolean actualizarUsuarioComun(UsuarioComun usuario) throws SQLException {
        // Actualiza los datos de un usuario de tipo UsuarioComun en la base de datos.
        // Devuelve un mensaje de error si la operación falla.
        // Devuelve SQLException si ocurre un error con la base de datos.
    }

    // Elimina un usuario de tipo UsuarioComun de la base de datos.
    // Devuelve un mensaje de error si la operación falla.
    // Devuelve SQLException si ocurre un error con la base de datos.
    public boolean eliminarUsuarioComun(UsuarioComun usuario) throws SQLException {
        // Elimina un usuario de tipo UsuarioComun de la base de datos.
        // Devuelve un mensaje de error si la operación falla.
        // Devuelve SQLException si ocurre un error con la base de datos.
    }

    // Devuelve un usuario de tipo UsuarioComun por su correo electrónico.
    // Devuelve null si no se encuentra el usuario.
    // Devuelve SQLException si ocurre un error con la base de datos.
    public UsuarioComun findByCorreo(String correo) throws SQLException {
        // Devuelve un usuario de tipo UsuarioComun por su correo electrónico.
        // Devuelve null si no se encuentra el usuario.
        // Devuelve SQLException si ocurre un error con la base de datos.
    }

    // Devuelve todos los usuarios de tipo UsuarioComun.
    // Devuelve una lista de usuarios de tipo UsuarioComun.
    // Devuelve SQLException si ocurre un error con la base de datos.
    public List<UsuarioComun> findAll() throws SQLException {
        // Devuelve todos los usuarios de tipo UsuarioComun.
        // Devuelve una lista de usuarios de tipo UsuarioComun.
        // Devuelve SQLException si ocurre un error con la base de datos.
    }

    // Verifica si los parámetros (correo y contraseña) son correctos para un usuario de tipo UsuarioComun.
    // Devuelve true si los parámetros son correctos.
    // Devuelve false si los parámetros no son correctos.
    // Devuelve SQLException si ocurre un error con la base de datos.
    public boolean check(String correo, String password) throws SQLException {
        // Verifica si los parámetros (correo y contraseña) son correctos para un usuario de tipo UsuarioComun.
        // Devuelve true si los parámetros son correctos.
        // Devuelve false si los parámetros no son correctos.
        // Devuelve SQLException si ocurre un error con la base de datos.
    }

    // Devuelve el número de comentarios de un usuario de tipo UsuarioComun.
    // Devuelve un mensaje de error si la operación falla.
    // Devuelve SQLException si ocurre un error con la base de datos.
    public int obtenerNumeroComentarios(UsuarioComun usuario) throws SQLException {
        // Devuelve el número de comentarios de un usuario de tipo UsuarioComun.
        // Devuelve un mensaje de error si la operación falla.
        // Devuelve SQLException si ocurre un error con la base de datos.
    }

    // Devuelve el nivel de participación de un usuario de tipo UsuarioComun.
    // Devuelve un mensaje de error si la operación falla.
    // Devuelve SQLException si ocurre un error con la base de datos.
    public int obtenerNivelParticipacion(UsuarioComun usuario) throws SQLException {
        // Devuelve el nivel de participación de un usuario de tipo UsuarioComun.
        // Devuelve un mensaje de error si la operación falla.
        // Devuelve SQLException si ocurre un error con la base de datos.
    }

    // Devuelve el nivel de participación de un usuario de tipo UsuarioComun.
    // Devuelve un mensaje de error si la operación falla.
    // Devuelve SQLException si ocurre un error con la base de datos.
    public String obtenerParticipacion(UsuarioComun usuario) throws SQLException {
        // Devuelve el nivel de participación de un usuario de tipo UsuarioComun.
        // Devuelve un mensaje de error si la operación falla.
        // Devuelve SQLException si ocurre un error con la base de datos.
    }
}
```



Por ejemplo, os voy a explicar cómo funciona el método updateParticipacion:

```
DAOUsuarioComun.java

/**
 * Actualiza el nivel de participación de un usuario de tipo Comun.
 *
 * @param usuarioComun El objeto UsuarioComun cuyo nivel de participación se actualizará.
 * @param participacion El objeto Participacion con el nuevo nivel de participación.
 * @return true si la actualización fue exitosa, false si no lo fue.
 * @throws SQLException Si ocurre un error con la base de datos.
 */
public boolean updateParticipacion(UsuarioComun usuarioComun, Participacion participacion) {
    boolean updated = false;

    try (Connection con = ConnectionBD.getConnection();
        PreparedStatement pst = con.prepareStatement(UPDATE_PARTICIPACION)) {

        // Establecer los parámetros
        pst.setString(1, participacion.toString());
        pst.setInt(2, usuarioComun.getIdUsuario());

        // Ejecutar la actualización y verificar el número de filas afectadas
        int rowsAffected = pst.executeUpdate();
        if (rowsAffected > 0) {
            updated = true; // Se actualizó al menos una fila
        }
    } catch (SQLException e) {
    }

    return updated;
}
```

Primero, el método llama a la instancia de conexión con getConnection. Luego se prepara la consulta y se insertan los valores necesarios, en este caso el nuevo nivel de participación, que es un valor de un enum Participacion (puede ser BAJA, MEDIA o ALTA).

A este método se le pasa tanto la participación como el objeto UsuarioComun que se quiere actualizar. El programa ejecuta la consulta, y si se ha actualizado correctamente, una

variable booleana llamada update pasa a true, y ese true es lo que se devuelve. Si no se actualiza nada, entonces salta una excepción.

## Mostrar

En mi programa hay dos listados que se muestran: uno de foros y otro de comentarios. Ahora os voy a explicar cómo funciona el mostrado de comentarios según el foro seleccionado (que se elige desde el otro listado).

El método cargarComentario funciona de la siguiente manera:

Se le pasa un objeto de tipo Foro. Dentro del método, se llama al DAO de texto

(daoTexto) y se llama el método findAllByForoId, al cual hay que enviarle el ID del foro (esto se hace con foro.getId\_foro()).

```
ForoControllerCreador.java

/**
 * Carga los comentarios asociados a un foro específico.
 *
 * @param foro Foro del cual se desean cargar los comentarios.
 * @throws SQLException si ocurre un error en la consulta.
 */
private void cargarComentarios(Foro foro) throws SQLException {
    DAOTexto daoTexto = new DAOTexto();
    List<Texto> comentarios = daoTexto.findAllByForoId(foro.getId_foro());
    listaComentarios.getItems().setAll(comentarios);
}
```

---

Este método `findAllByForoId` se encarga de devolver una lista con todos los textos que tengan el ID de ese foro. Esto es posible porque al registrar un comentario, se guarda también el `id_foro` para saber a qué foro pertenece.

Una vez obtenida la lista de textos, la asignamos a una lista de tipo `Texto`, y luego cargamos esos comentarios en nuestro `ListView` llamado `listaComentarios`.

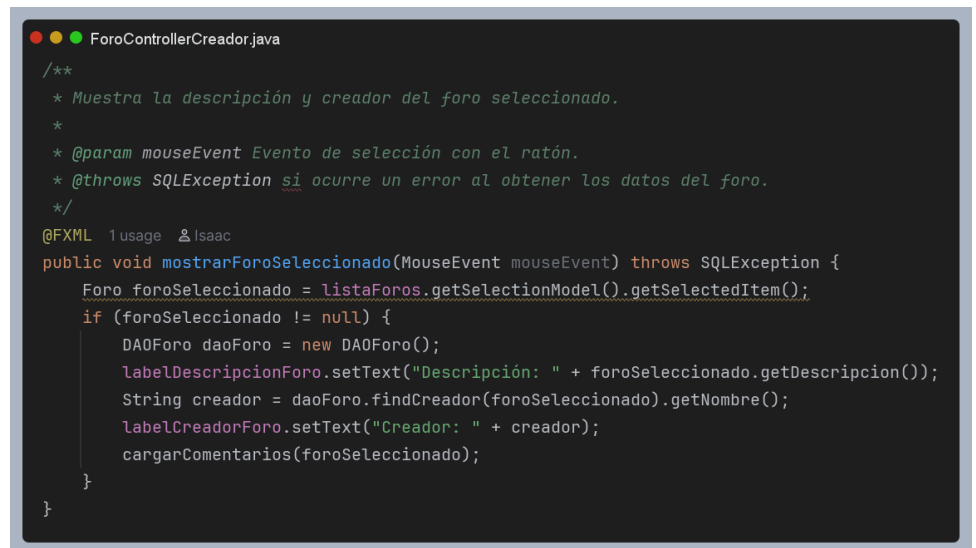
Ahora bien, ¿cómo sabe el programa cuándo debe ejecutarse `cargarComentario`?

Pues hay otro método llamado `mostrarForoSeleccionado`, que es un evento `MouseEvent`. Es decir, cuando se selecciona un foro en la lista, ese foro se guarda en un objeto llamado `foroSeleccionado`, y este se pasa al método `cargarComentario` para que aparezcan los comentarios correspondientes.

Además, el método `mostrarForoSeleccionado` también muestra información del foro: la descripción y el nombre del creador.

- La descripción se muestra con un `setText()` en el `Label` correspondiente.
- El nombre del creador se obtiene llamando a un método del `daoForo` llamado

`findCreador`, al que se le pasa el foro seleccionado y este nos devuelve el objeto creador. A ese objeto le hacemos un `getNombre()` y ya tenemos el nombre del creador.



```

// **
// * Muestra la descripción y creador del foro seleccionado.
// *
// * @param mouseEvent Evento de selección con el ratón.
// * @throws SQLException si ocurre un error al obtener los datos del foro.
// */
@FXML 1 usage 2 Isaac
public void mostrarForoSeleccionado(MouseEvent mouseEvent) throws SQLException {
    Foro foroSeleccionado = listaForos.getSelectionModel().getSelectedItem();
    if (foroSeleccionado != null) {
        DAOForo daoForo = new DAOForo();
        labelDescripcionForo.setText("Descripción: " + foroSeleccionado.getDescripcion());
        String creador = daoForo.findCreador(foroSeleccionado).getNombre();
        labelCreadorForo.setText("Creador: " + creador);
        cargarComentarios(foroSeleccionado);
    }
}

```

## Insertar

En el programa se puede insertar usuario o foro. Os voy a explicar cómo se inserta un foro en mi aplicación.

Primero, en la clase Foro tenemos dos vistas: foroCreador o foroComun.

Para añadir un foro deberemos ser creadores, por lo tanto, en foroCreador hay un método llamado `añadirNuevoForo`, donde nos abrirá la pestaña fija `nuevoForo.fxml`, la cual está conectada a `CrearForoController`.

```

    ForoControllerCreador.java
    /**
     * Abre una ventana para añadir un nuevo foro.
     *
     * @param actionEvent Evento del botón.
     * @throws IOException si falla la carga del FXML.
     */
    @FXML 1 usage 2 Isaac
    public void añadirNuevoForo(ActionEvent actionEvent) throws IOException {
        FXMLLoader fxmlLoader = new FXMLLoader(Start.class.getResource("nuevoForo.fxml"));
        Scene scene = new Scene(fxmlLoader.load());
        Stage stage = new Stage();
        stage.setScene(scene);
        stage.setResizable(false);
        stage.setTitle("Crear Foro");
        stage.showAndWait();
        actualizarListaForos();
    }

```

En esa vista insertaremos el título y la descripción del foro (es obligatorio llenar los dos campos), y luego, cuando pulsemos el botón Añadir, se nos ejecutará el método `crearForo()`.

```

    CrearForoController.java
    /**
     * Método llamado cuando el usuario hace clic en el botón para crear un foro.
     * Este método verifica que los campos no estén vacíos, crea el foro en la base de datos
     * y cierra la ventana de creación del foro.
     *
     * @throws SQLException si ocurre un error al interactuar con la base de datos
     */
    @FXML 1 Isaac
    private void crearForo() throws SQLException {
        String titulo = tituloField.getText().trim();
        String descripcion = descripcionField.getText().trim();

        if (titulo.isEmpty() || descripcion.isEmpty()) {
            mensajeLabel.setText("Todos los campos son obligatorios.");
            return;
        }

        Usuario usuarioActual = SesionUsuario.getUsuario();

        if (usuarioActual == null) {
            mensajeLabel.setText("Error: No hay un usuario logueado.");
            return;
        }
        System.out.println(usuarioActual.getTipoUsuario());

        UsuarioCreador creador = (UsuarioCreador) usuarioActual;
        creador.setNum_ForosCreados();
        DAOUsuarioCreador daoUsuarioCreador = new DAOUsuarioCreador();
        daoUsuarioCreador.updateNumForos(creador);

        Foro nuevoForo = new Foro();
        nuevoForo.setTitulo(titulo);
        nuevoForo.setDescripcion(descripcion);
        nuevoForo.setId_creador(usuarioActual.getId_Usuario());

        DAOForo daoForo = new DAOForo();
        daoForo.insert(nuevoForo, usuarioActual);

        Stage stage = (Stage) btnCrearForo.getScene().getWindow();
        stage.close();
    }

```

Ese método recogerá el título y la descripción escrita con un `getText()`. También el código nos comprueba que no haya ningún campo vacío, y que exista un usuario logueado.

Luego, como existe una clase `SesionUsuario` que crea un usuario que es el que está logueado, por ese usuario lo cambiamos a **\*\*usuario creador\*\*** para poder usar los métodos del DAO.

Una vez cambiado, usamos el método `setNumForo` para que nos ponga el número de foros +1, y después usamos `dao.updateNumForos`, que lo que le

enviaremos a este método es el usuario creador, y ese método cogerá el ID de ese usuario y el número de foros, y realizará un `update`.

Este método nos devuelve un `true` si se ha actualizado, y `false` si no se ha actualizado.

Una vez haya devuelto `true` o `false`, el método de `crearForo` seguirá. Creamos un foro con los datos ingresados en la interfaz gráfica, y este foro lo usamos en un método `insert`, que le envía el foro y el creador.

```
DAOUsuarioCreador.java
/**
 * Actualiza el número de foros creados por un usuario creador.
 *
 * @param creador El usuario creador cuya cantidad de foros creados se actualizará.
 * @return true si la actualización fue exitosa, false en caso contrario.
 * @throws SQLException si ocurre un error de base de datos.
 */
public boolean updateNumForos (UsuarioCreador creador) throws SQLException {
    Boolean update = false;
    Connection con = ConnectionBD.getConnection();
    try (PreparedStatement pst = con.prepareStatement(UPDATE_NUM_FOROS)) {
        pst.setInt( parameterIndex:1, creador.getNum_ForosCreados());
        pst.setInt( parameterIndex:2, creador.getId_Usuario());
        pst.executeUpdate();
        update = true;
    } catch (SQLException e) {
        throw new RuntimeException(e);
    }
    return update;
}
```

```
DAOForo.java
/**
 * Inserta un nuevo foro en la base de datos.
 *
 * @param foro Foro a insertar.
 * @param creador Usuario que crea el foro.
 * @return Foro insertado con su ID generado.
 * @throws SQLException si ocurre un error en la base de datos.
 */
public Foro insert(Foro foro, Usuario creador) throws SQLException {
    if (foro != null && creador != null) {
        try (PreparedStatement pst = ConnectionBD.getConnection().prepareStatement(INSERT_FORO, Statement.RETURN_GENERATED_KEYS)) {
            pst.setString( parameterIndex:1, foro.getTitulo());
            pst.setString( parameterIndex:2, foro.getDescripcion());
            pst.setDate( parameterIndex:3, foro.getFecha_creacion());
            pst.setInt( parameterIndex:4, creador.getId_Usuario());

            pst.executeUpdate();
            ResultSet rs = pst.getGeneratedKeys();
            if (rs.next()) {
                foro.setId_foro(rs.getInt( columnIndex:1));
                foro.setId_creador(creador.getId_Usuario());
            }
        }
    }
    return foro;
}
```

Este método inserta el título y la descripción del foro, y le asigna a la ID del foro el usuario que se ha enviado al método.

Una vez terminado el método de crear, volverá a `añadirForo`, el método del controller, y actualiza la lista de foros con el método del creador, el cual devuelve un listado si el usuario

actual existe y enviará únicamente los foros de ese usuario actual con un `findForosByID`, que funciona enviándole el ID de usuario, y este hace una consulta `SELECT` y nos devuelve solo una lista de foros de esa ID.

## Eliminar

En el programa solo podrá eliminar los creados, es decir, los comunes se es el que publica el comentario no podrá borrarlo. Los creadores tienen un botón eliminar que elimina tanto comentarios como foros, dependiendo lo que tenga seleccionado. Funciona de la siguiente manera:

Primero detecta a través de una variable si se ha seleccionado un foro o un texto. Pongamos el ejemplo que se ha seleccionado el texto. Pues se creará un DAOTexto el cual llamará la función delete, la cual le enviamos el comentario seleccionado. Dentro del método eliminar comprobará si el comentario tiene contenido. Si

```

// ForoControllerCreador.java
/**
 * Elimina el foro o comentario seleccionado por el usuario.
 *
 * Este método verifica si el usuario ha seleccionado un foro o comentario.
 * Si el usuario ha seleccionado un comentario, se eliminará este comentario.
 * Si ha seleccionado un foro, se verificará que el usuario que intenta eliminar el foro sea el creador del mismo.
 * Si es el creador, el foro se eliminará. De lo contrario, se mostrará un mensaje de error.
 *
 * @throws SQLException si ocurre un error durante la eliminación en la base de datos.
 */
@FXML @Isaac
private void eliminarItemSeleccionado() throws SQLException {
    Foro foroSeleccionado = listaForos.getSelectionModel().getSelectedItem();
    Texto comentarioSeleccionado = listaComentarios.getSelectionModel().getSelectedItem();

    if (comentarioSeleccionado != null) {
        DAOTexto daoTexto = new DAOTexto();
        boolean eliminado = daoTexto.delete(comentarioSeleccionado);
        if (eliminado) {
            mensajeAlerta.setText("Comentario eliminado con éxito.");
            cargarComentarios(foroSeleccionado);
        } else {
            mensajeAlerta.setText("No se pudo eliminar el comentario.");
        }
        return;
    }

    if (foroSeleccionado != null) {
        Usuario usuarioActual = SesionUsuario.getUsuario();

        if (usuarioActual != null && foroSeleccionado.getId_creador() == usuarioActual.getId_usuario()) {
            DAOForo daoForo = new DAOForo();
            boolean eliminado = daoForo.delete(foroSeleccionado);

            if (eliminado) {
                mensajeAlerta.setText("Foro eliminado con éxito.");
                cargarForos();
                listaComentarios.getItems().clear();
                labelDescripcionForo.setText("Descripción:");
                labelCreadorForo.setText("Creador:");
            } else {
                mensajeAlerta.setText("No se pudo eliminar el foro.");
            }
        } else {
            mensajeAlerta.setText("Tienes que ser propietario del foro para eliminarlo.");
        }
    } else {
        mensajeAlerta.setText("Debes seleccionar un foro o comentario para eliminarlo.");
    }
}

```

```

// ForoControllerCreador.java
/**
 * Carga los comentarios asociados a un foro específico.
 *
 * @param foro Foro del cual se desean cargar los comentarios.
 * @throws SQLException si ocurre un error en la consulta.
 */
private void cargarComentarios(Foro foro) throws SQLException {
    DAOTexto daoTexto = new DAOTexto();
    List<Texto> comentarios = daoTexto.findAllByForoId(foro.getId_foro());
    listaComentarios.getItems().setAll(comentarios);
}

```

no tiene, nos saltará una excepción. Si no, haremos un getConnection de instancia y enviaremos por parámetros el ID del texto que hemos seleccionado, y nos devolverá true si ha sido eliminado o false si no. Si es true, entrará en la condición y cargará de

```

// DAOTexto.java
/**
 * Elimina un texto de la base de datos.
 *
 * @param texto El objeto Texto que se desea eliminar.
 * @return true si el texto se elimina correctamente, false si no.
 * @throws SQLException Si ocurre un error con la base de datos.
 */
public boolean delete(Texto texto) throws SQLException {
    boolean deleted = false;

    if (texto == null || texto.getId_contenido() == 0) {
        throw new IllegalArgumentException("Texto inválido o sin ID.");
    }

    try {
        Connection conn = ConnectionBD.getConnection();
        PreparedStatement pst = conn.prepareStatement(DELETE_TEXTO);

        pst.setInt(parameterIndex: 1, texto.getId_contenido());
        int rowsAffected = pst.executeUpdate();
        deleted = rowsAffected > 0;
    }

    return deleted;
}

```

nuevos comentarios, utilizando un método parecido a cargarForo. Enviaremos el foro al método y con un findAllIDForo, que nos devuelve la lista de foros de esa ID, lo asignamos a una lista de comentarios y esta la imprimimos en el listado ListView.

## Actualizar

Mi aplicación nos permite actualizar foros, solo a los creados y solo a sus foros creados. Se actualiza un foro de la siguiente forma: primeramente pulsaremos el botón de actualizar. Se nos abrirá una pestaña emergente fija y en ella nos aparecerá un desplegable con los foros únicamente del creador registrado, esto se consigue con el método cargarForo explicado anteriormente.

```

// ForoControllerCreador.java
/**
 * Abre una ventana para actualizar un foro.
 */
@param actionEvent Evento del botón.
* @throws IOException si falla la carga del FXML.
*/
@FXML @Isaac
public void actualizarForo(ActionEvent actionEvent) throws IOException {
    FXMLLoader fxmlLoader = new FXMLLoader(Start.class.getResource("actualizarforo.fxml"));
    Scene scene = new Scene(fxmlLoader.load());
    Stage stage = new Stage();
    stage.setScene(scene);
    stage.setResizable(false);
    stage.setTitle("Actualizar Foro");
    stage.showAndWait();
    actualizarListaForos();
}

```

```

// ActualizarForoController.java
@FXML @Isaac
private void actualizarForo() throws SQLException {
    String titulo = nombreForo.getText().trim(); // Obtiene el título introducido
    String descripcion = descripcionActualizar.getText().trim(); // Obtiene la descripción introducida

    Usuario usuarioActual = SesionUsuario.getUsuario();

    if (usuarioActual == null) {
        mensajeLabel.setText("Error: No hay un usuario logueado.");
        return;
    }

    Foro foroSeleccionado = listaForos.getSelectionModel().getSelectedItem();

    if (foroSeleccionado == null) {
        mensajeLabel.setText("Debe seleccionar un foro para actualizar.");
        return;
    }

    Foro nuevoForo = new Foro();
    if (!titulo.isEmpty()) {
        nuevoForo.setTitulo(titulo);
    } else {
        nuevoForo.setTitulo(foroSeleccionado.getTitulo());
    }

    if (!descripcion.isEmpty()) {
        nuevoForo.setDescripcion(descripcion);
    } else {
        nuevoForo.setDescripcion(foroSeleccionado.getDescripcion());
    }

    DAOForo daoForo = new DAOForo();

    boolean actualizado = daoForo.update(nuevoForo, foroSeleccionado, usuarioActual);

    if (actualizado) {
        mensajeLabel.setText("Foro actualizado con éxito.");

        Stage stage = (Stage) btnActualizarForo.getScene().getWindow();
        stage.close();
    } else {
        mensajeLabel.setText("No se pudo actualizar el foro.");
    }
}

```

Rellenamos el título y la descripción, no es necesario rellenar las dos. Crearemos un nuevo objeto Foro, luego detecta si hay algún campo vacío; si es así, le asignará al nuevo foro el contenido del foro viejo y si no, lo actualizará. Una vez creado a la perfección el foro nuevo, llamaremos a través de DAOForo a updateForo, al cual le envío el nuevoForo, el viejo y el usuario que está logueado.

En ese método, primero comprobará que esté todo

relleno, es decir, que no sea null, y luego encuentra los campos de título y descripción del foro. Enviamos el ID del foro viejo y el ID del usuario. Nos devolverá true si se actualiza y false si no. Una vez devuelto, se cerrará la ventana de actualizar y se actualizará la lista con actualizarListaForos.

```
DAOForo.java

/**
 * Actualiza un foro existente si el usuario es su creador.
 *
 * @param foroNuevo Foro con los datos actualizados.
 * @param foroViejo Foro original a actualizar.
 * @param creador Usuario que realiza la modificación.
 * @return true si la actualización fue exitosa, false en caso contrario.
 * @throws SQLException si ocurre un error en la base de datos.
 */
public boolean update(Foro foroNuevo, Foro foroViejo, Usuario creador) throws SQLException {
    boolean actualizado = false;
    if (foroNuevo != null && foroViejo != null && creador != null && foroViejo.getId_creador() == creador.getId_Usuario()) {
        try (PreparedStatement pst = ConnectionBD.getConnection().prepareStatement(UPDATE_FORO)) {
            pst.setString(parameterIndex: 1, foroNuevo.getTitulo());
            pst.setString(parameterIndex: 2, foroNuevo.getDescripcion());
            pst.setInt(parameterIndex: 3, foroViejo.getId_foro());
            pst.setInt(parameterIndex: 4, creador.getId_Usuario());

            int filasAfectadas = pst.executeUpdate();
            actualizado = filasAfectadas > 0;
        }
    }
    return actualizado;
}
```