

DEPARTAMENTO DE INFORMÁTICA Y COMUNICACIONES

Evaluación Conectores Proyecto Gestión FCT



CICLO: 2 DAM

Alumnos: Daniel Darmanin Casariego, Cristian Quintero

CURSO: 2024 / 2025

Índice

Evaluación Conectores Proyecto Gestión FCT.....	1
Enunciado.....	3
Desarrollo de la actividad.....	5
Resumen.....	5
Definición del Proyecto.....	5
Objetivo.....	5
Funcionalidades Principales.....	5
Requerimientos Técnicos.....	6
Informe Técnico.....	6
Beneficios.....	6
Apartados.....	7
1: Modelo Entidad-Relación.....	7
2: Implementación de la Base de Datos.....	8
3. Estructura de la Base de Datos.....	9
4: Interfaz de usuario (FXML).....	12
4.1: Código de la interfaz.....	16
5: Conexión HikariCP.....	25
6. Tecnologías Utilizadas.....	27
1.-Lenguaje de Programación:.....	27
2.-Frameworks:.....	28
3.- Base de datos:.....	28
4.-Gestión de Conexión:.....	28
Bibliografía.....	28
Conclusión.....	28

Enunciado

En una institución educativa que gestiona prácticas profesionales de sus alumnos en diversas empresas, surge la necesidad de contar con un sistema que centralice y organice la información relacionada con estos programas de prácticas. La institución requiere un sistema que permita registrar y gestionar tanto los detalles de las empresas colaboradoras como la información de cada alumno, los tutores asignados, las visitas de supervisión, los comentarios de seguimiento de cada alumno y otros aspectos críticos que aseguren un adecuado seguimiento y evaluación de las prácticas.

1. Gestión de Empresas:
 - Registro de empresas que participan en el programa de prácticas, incluyendo información de contacto y datos básicos de la organización.
 - Gestión de contactos y tutores de empresa asignados a los alumnos en prácticas.
 - Registro de comentarios y observaciones durante el proceso de captación de empresas, con la finalidad de documentar la evaluación y el interés de la empresa en participar en el programa de prácticas.
2. Gestión de Alumnos:
 - Registro de alumnos participantes en el programa de prácticas.
 - Registro del programa o carrera del alumno, datos de contacto y otros detalles necesarios para su seguimiento.
3. Asignación de Prácticas:
 - Registro de asignaciones de prácticas de alumnos a empresas específicas.
 - Asignación de tutores de empresa y tutores docentes a cada alumno en práctica.
 - Control de plazas disponibles en las empresas, categorizadas por especialidad.
4. Gestión de Visitas y Seguimiento:
 - Registro de visitas de seguimiento realizadas por tutores docentes a los alumnos en sus empresas asignadas.
 - Almacenamiento de observaciones y comentarios realizados por los tutores docentes sobre el progreso de los alumnos en sus prácticas.
5. Gestión de Comentarios durante la Captación de Empresas:
 - Registro de comentarios y notas adicionales durante el proceso de captación de empresas interesadas en el programa.

- Seguimiento de la evolución de la captación de empresas, para evaluar su perfil y condiciones de colaboración.

6. Sistema CRUD (Crear, Leer, Actualizar, Eliminar):

- Implementación de operaciones CRUD para cada una de las entidades en el sistema: Alumnos, Empresas, Tutores, Asignaciones y Comentarios.
- Facilitar la búsqueda, actualización y eliminación de datos cuando sea necesario.

7. Generación del Modelo Entidad-Relación (E-R):

- El sistema debe contar con un modelo Entidad-Relación (E-R) que detalle las relaciones entre las entidades involucradas, así como los atributos clave de cada una de ellas.
- El modelo E-R debe ilustrar claramente las conexiones entre empresas, alumnos, asignaciones, tutores, visitas y comentarios, sirviendo como base para la creación de la base de datos y su posterior implementación.

8. Informe Técnico del Sistema:

- Elaboración de un informe técnico que detalle la arquitectura del sistema, las decisiones de diseño, las tecnologías utilizadas y las configuraciones aplicadas en el sistema.
- El informe debe incluir una descripción detallada de cada funcionalidad, el modelo E-R, la estructura de la base de datos, y las configuraciones de conexión a la base de datos (incluyendo el uso del pool de conexiones HikariCP).
- Incluir detalles sobre la implementación del CRUD, las interfaces de usuario en Java Swing, y cómo se realizó la conexión entre la aplicación y la base de datos.

9. Interfaz de Usuario Opcional (Java Swing o JavaFX):

- La implementación de una interfaz gráfica de usuario (GUI) es opcional, pero su inclusión en el sistema mejorará el puntaje final obtenido por el alumno.
- Esta interfaz debe ser intuitiva y permitir a los usuarios realizar las operaciones CRUD de forma fácil y eficiente, así como visualizar los registros y realizar búsquedas sobre los datos almacenados en la base de datos.

Desarrollo de la actividad

Resumen

Definición del Proyecto

Este proyecto se llevará a cabo en tres fases:

1. **Primera Fase:** Definición del proyecto, incluyendo objetivos, justificación y descripción del problema.
2. **Segunda Fase:** Diseño del sistema de información, incluyendo la creación de un procedimiento, selección de herramientas para pruebas, ejecución del sistema y análisis de resultados.
3. **Tercera Fase:** Interpretación de resultados y generación de recomendaciones.

Objetivo

Desarrollar un sistema para gestionar prácticas profesionales de alumnos en empresas colaboradoras, centralizando y organizando datos relacionados con alumnos, empresas, tutores, visitas y comentarios de seguimiento. Esto permitirá un mejor control y evaluación de las prácticas, mejorando la transparencia y el acceso a la información.

Funcionalidades Principales

- **Gestión de Empresas:** Registro de empresas, contactos y tutores asignados, junto con observaciones sobre el proceso de captación.
- **Gestión de Alumnos:** Registro de alumnos, programas de estudio y detalles de contacto.
- **Asignación de Prácticas:** Vinculación de alumnos con empresas, asignación de tutores y control de plazas disponibles.
- **Seguimiento y Visitas:** Registro de visitas de seguimiento, observaciones de tutores y progreso del alumno.
- **Sistema CRUD:** Crear, leer, actualizar y eliminar datos de las entidades principales del sistema (alumnos, empresas, tutores, asignaciones, etc.).
- **Modelo Entidad-Relación (E-R):** Representación clara de las relaciones entre empresas, alumnos, asignaciones, tutores, visitas y comentarios.

Requerimientos Técnicos

- **Base de Datos Relacional:** Almacenar información sobre empresas, alumnos, asignaciones, visitas y comentarios.
- **Pool de Conexiones HikariCP:** Mejora de eficiencia en operaciones concurrentes con la base de datos.
- **Interfaz de Usuario (opcional):** Implementación en Java Swing o JavaFX para facilitar el uso del sistema.

Informe Técnico

- Descripción de funcionalidades, modelo E-R y arquitectura del sistema.
- Detalles sobre el CRUD y configuración de base de datos.
- Explicación de la conexión con el pool de HikariCP y posibles interfaces gráficas.

Beneficios

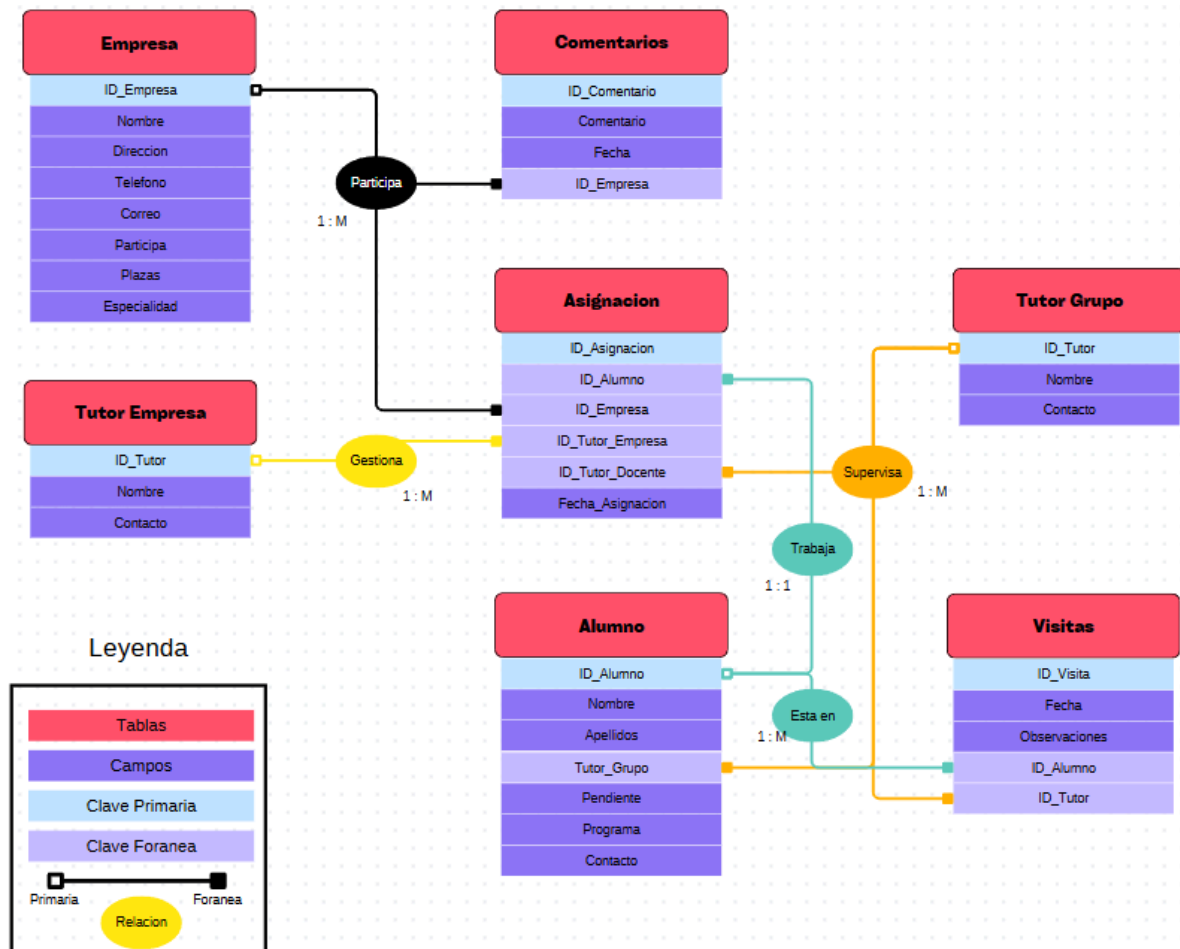
- **Eficiencia:** Simplificación del acceso y gestión de datos.
- **Control Mejorado:** Seguimiento detallado del progreso de alumnos y condiciones de empresas colaboradoras.
- **Optimización:** Reducción de tiempos de respuesta en la gestión de datos.

Este sistema busca modernizar y optimizar la gestión de prácticas profesionales, beneficiando tanto a la institución educativa como a los alumnos y empresas colaboradoras.

Apartados

1: Modelo Entidad-Relación

Para la realización de este proyecto es necesaria la creación de una base de datos relacional que se encargue de manejar los datos que nos piden. Con este fin hemos creado una base de datos con las siguientes tablas, campos y relaciones:



2: Implementación de la Base de Datos

Para crear la base de datos que almacena nuestros datos hemos utilizado el sistema de xampp con **MySQL**. El motivo del uso de este sistema de base de datos ha sido por la usabilidad y seguridad que este sistema proporciona. **MySQL** es altamente eficiente en el manejo de grandes volúmenes de datos y múltiples usuarios concurrentes, ideal para gestionar información de alumnos, empresas, tutores y asignaciones. Su sintaxis SQL es sencilla y ampliamente conocida, lo que facilita su aprendizaje y la incorporación de nuevos desarrolladores al proyecto. Proporciona mecanismos para proteger los datos, como cifrado, gestión de usuarios y permisos, además de soporte para replicación, garantizando la disponibilidad del sistema. Al ser una tecnología ampliamente adoptada, ofrece acceso a una gran comunidad de soporte, documentación oficial extensa y numerosos recursos educativos. **MySQL** maneja relaciones entre tablas con claves primarias y foráneas, facilitando la representación de datos complejos, como la asociación entre estudiantes, empresas y tutores.

Por su rendimiento, escalabilidad, facilidad de integración y características de seguridad, **MySQL** es una opción adecuada y eficiente para un proyecto de gestión de prácticas FCT, permitiendo un manejo efectivo de la información clave del sistema.

Para que la aplicación tenga acceso a la base de datos se creó una clase con un script **SQL** que solo se ejecuta cuando la base de datos no existe.

La siguiente imagen muestra el principio del script.

```
package dad.Conexion;

import ...

public class DatabaseInitializer { no usages

    private static final String SCRIPT_SQL = """ 1 usage

    CREATE TABLE 'alumno' (
        'Id_Alumno' int(11) NOT NULL,
        'Nombre' varchar(25) NOT NULL,
        'Apellidos' varchar(35) NOT NULL,
        'Tutor_Grupo' int(11) DEFAULT NULL,
        'Pendiente' tinyint(1) DEFAULT 0,
        'Programa' varchar(50) DEFAULT NULL,
        'Contacto' varchar(50) DEFAULT NULL
    ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_spanish_ci;

    CREATE TABLE 'alumnos_empresas_rel' (
        'Id_Asignacion' int(11) NOT NULL,
        'Id_Alumno' int(11) NOT NULL,
        'Nombre' varchar(50) COLLATE utf8mb4_spanish_ci NOT NULL,
        'Apellidos' varchar(90) COLLATE utf8mb4_spanish_ci NOT NULL,
        'Id_Empresa' int(11) NOT NULL,
        'Id_Tutor_Empresa' int(11) DEFAULT NULL,
        'Id_Tutor_Docente' int(11) DEFAULT NULL,
        'Fecha_Inicio' date NOT NULL,
        'Fecha_Fin' date NOT NULL
    ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_spanish_ci;

    CREATE TABLE 'asignaciones_empresas' (
```


Mientras que está muestra la ejecución.

```
public static void initializeDatabase() throws SQLException { 1 usage
    try (Connection connection = Conectar.getConnection();
        Statement statement = connection.createStatement()) {

        for (String query : SCRIPT_SQL.split( regex: ";" )) {
            if (!query.trim().isEmpty()) {
                statement.execute( sql: query.trim() + ";" );
            }
        }

        System.out.println("Base de datos inicializada con éxito.");

    } catch (SQLException e) {
        throw new SQLException("Error ejecutando el script SQL", e);
    }
}
```

3. Estructura de la Base de Datos

La estructura de la base de datos está formada por las siguientes tablas:

- Tabla: Alumno

Columnas:

- **Id_Alumno** (int(11)): Identificador único para cada alumno. Es la clave primaria y se autoincrementa automáticamente.
- **Nombre** (varchar(25)): Nombre del alumno. No puede ser nulo.
- **Apellidos** (varchar(35)): Apellidos del alumno. No puede ser nulo.
- **Tutor_Grupo** (int(11)): Identificador del tutor del grupo asignado al alumno. Es una clave foránea que hace referencia a la tabla "tutor_grupo"
- **nombreTutor** (varchar(80)): Nombre del tutor asignado al alumno. No puede ser nulo.
- **Pendiente** (tinyint(1)): Indica si el alumno tiene alguna tarea pendiente (valor booleano, 0 o 1).
- **Curso** (varchar(50)): Nombre del curso en el que está matriculado el alumno (por ejemplo, "Segundo DAM").
- **Contacto** (varchar(50)): Información de contacto del alumno (puede ser un correo electrónico o número de teléfono).

- Tabla: Alumnos_Empresas_Rel

Columnas:

- **Id_Asignacion** (int(11)): Identificador único de la asignación entre un alumno y una empresa. Clave primaria que se autoincrementa.
- **Id_Alumno** (int(11)): Identificador del alumno. Clave foránea que hace referencia a la tabla alumno.
- **Nombre** (varchar(50)): Nombre del alumno asignado a la empresa.
- **Apellidos** (varchar(90)): Apellidos del alumno asignado a la empresa.
- **Id_Empresa** (int(11)): Identificador de la empresa asignada al alumno. Clave foránea que hace referencia a la tabla empresas.
- **Id_Tutor_Empresa** (int(11)): Identificador del tutor de la empresa asignado al alumno. Puede ser nulo.
- **Empresa** (varchar(100)): Nombre de la empresa asignada al alumno.
- **Id_Tutor_Docente** (int(11)): Identificador del tutor docente que supervisa al alumno. Puede ser nulo.
- **Fecha_Inicio** (date): Fecha de inicio de la asignación del alumno a la empresa.
- **Fecha_Fin** (date): Fecha de finalización de la asignación del alumno a la empresa.

- Tabla: Comentarios_Empresa

Columnas:

- **Id_Comentario** (int(11)): Identificador único del comentario. Clave primaria.
- **Comentario** (text): Texto del comentario sobre la empresa.
- **Fecha_Comentario** (timestamp): Fecha en que se realizó el comentario.
- **Id_Empresa** (int(11)): Identificador de la empresa comentada. Clave foránea que hace referencia a la tabla empresas.
- **nombreEmpresa** (varchar(80)): Nombre de la empresa comentada.

- Tabla: Empresas

Columnas:

- **Id_Empresa** (int(11)): Identificador único de la empresa. Clave primaria.
- **Nombre** (varchar(100)): Nombre de la empresa.
- **Direccion** (text): Dirección de la empresa.
- **Telefono** (varchar(15)): Teléfono de la empresa.
- **Correo** (varchar(100)): Correo de contacto de la empresa.
- **Participa** (tinyint(1)): Indica si la empresa participa en el sistema (0 o 1).
- **Plazas** (int(11)): Número de plazas disponibles para alumnos.
- **Especialidad** (varchar(50)): Especialidad de la empresa.

- Tabla: Tutor_Grupo:

Columnas:

- **Id_Tutor** (int(11)): Identificador único del tutor del grupo. Clave primaria.
- **Nombre** (varchar(35)): Nombre del tutor del grupo.
- **Grupo** (varchar(32)): Nombre del grupo que el tutor supervisa.

- Tabla: Visitas:

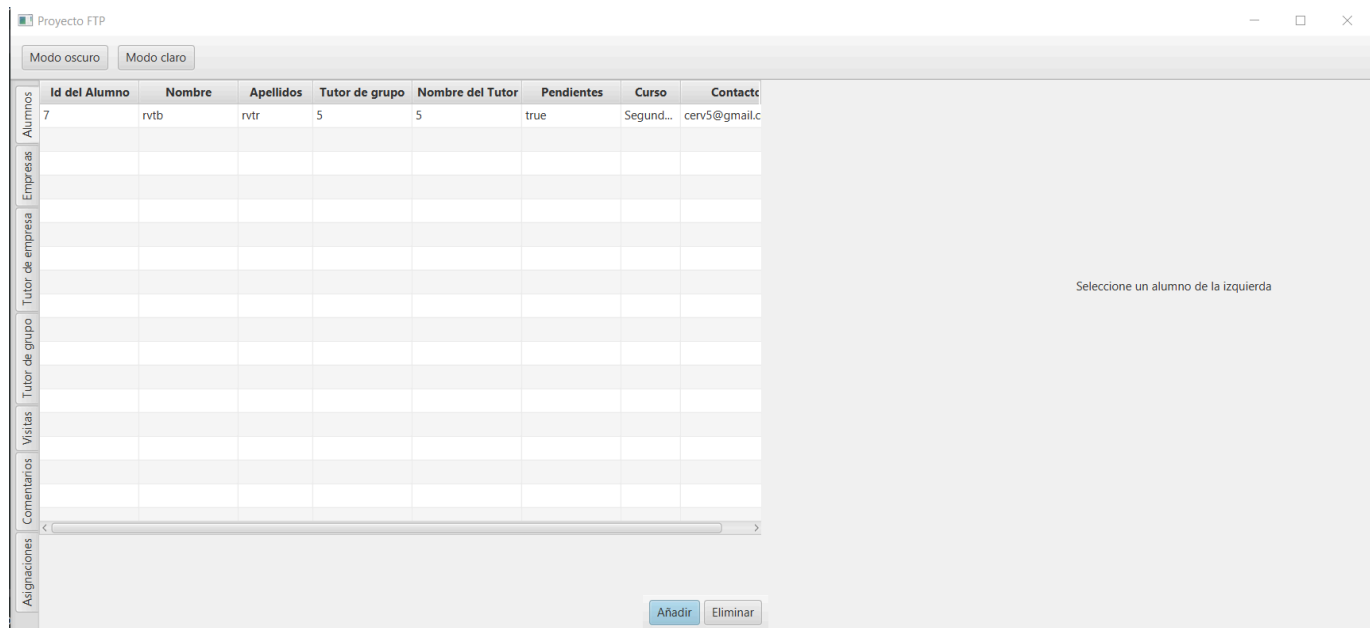
Columnas:

- **Id_Visita** (int(11)): Identificador único de la visita. Clave primaria.
- **Fecha_Visita** (date): Fecha de la visita realizada.
- **Observaciones** (text): Observaciones sobre la visita.
- **Id_Alumno** (int(11)): Identificador del alumno que fue visitado. Clave foránea que hace referencia a la tabla **Alumno**.
- **nombreAlumno** (varchar(80)): Nombre del alumno visitado.
- **apellidosAlumno** (varchar(100)): Apellidos del alumno visitado.
- **Id_Tutor** (int(11)): Identificador del tutor del grupo que realizó la visita. Clave foránea que hace referencia a la tabla **Tutor_Grupo**.
- **nombreTutorGrupo** (varchar(80)): Nombre del tutor del grupo que realizó la visita.

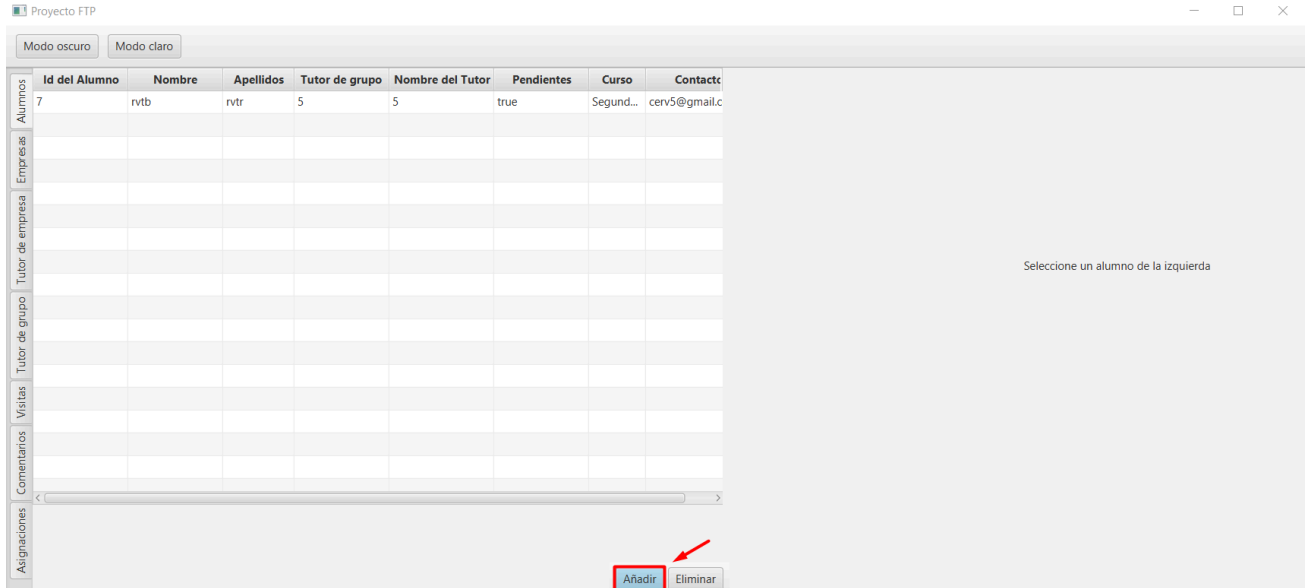
4: Interfaz de usuario (FXML)

Para interactuar fácilmente con esta base de datos se implementó una interfaz de usuario creada con javaFX que incluye las funciones necesarias para el manejo de datos: inserción, lectura, modificación y eliminación. Para un uso intuitivo y sencillo se usaron varias pestañas que interactúan con una tabla diferente cada una, la pestaña alumno interactúa con la tabla alumno, etc.

Lo primero que verá el usuario será esta ventana:



Si el usuario desea añadir un nuevo alumno en la tabla “Alumno” deberá pulsar el botón “Añadir”



Proyecto FTP

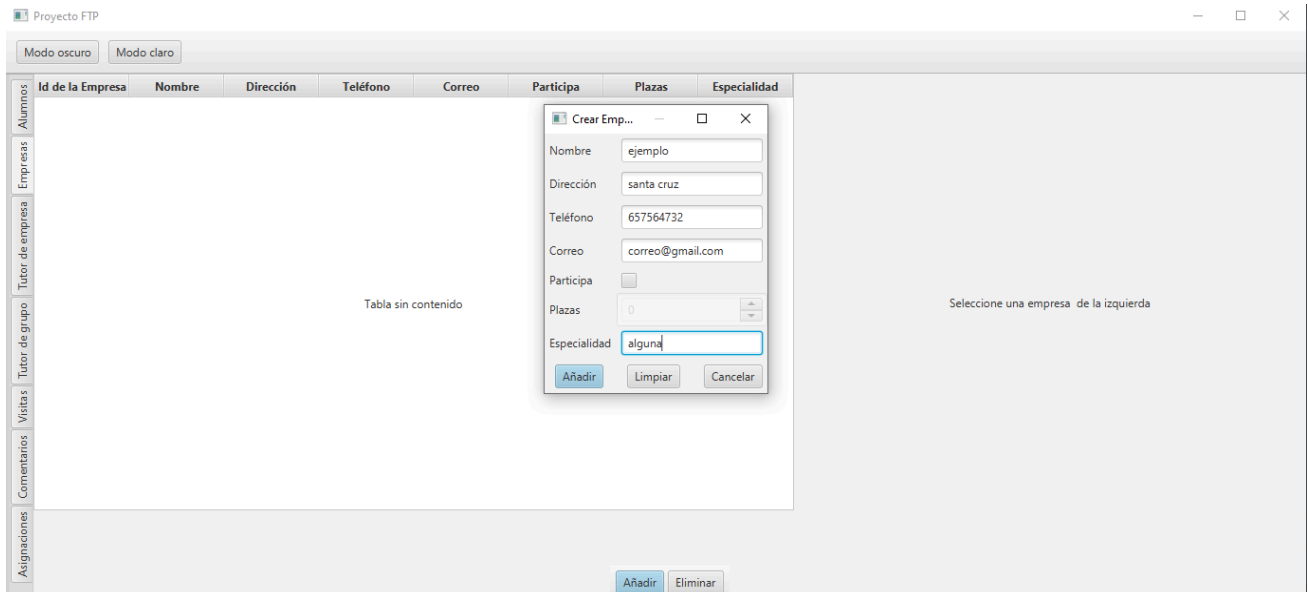
Modo oscuro Modo claro

	Id del Alumno	Nombre	Apellidos	Tutor de grupo	Nombre del Tutor	Pendientes	Curso	Contacto
Alumnos	7	rvtb	rvtr	5	5	true	Segund...	cerv5@gmail.c
Empresas								
Tutor de empresa								
Tutor de grupo								
Visitas								
Comentarios								
Asignaciones								

Seleccione un alumno de la izquierda

Añadir Eliminar

Y aparecerá esta ventana:



Proyecto FTP

Modo oscuro Modo claro

	Id de la Empresa	Nombre	Dirección	Teléfono	Correo	Participa	Plazas	Especialidad
Alumnos								
Empresas								
Tutor de empresa								
Tutor de grupo								
Visitas								
Comentarios								
Asignaciones								

Tabla sin contenido

Seleccione una empresa de la izquierda

Crear Emp...

Nombre: ejemplo

Dirección: santa cruz

Teléfono: 657564732

Correo: correo@gmail.com

Participa: ☐

Plazas: 0

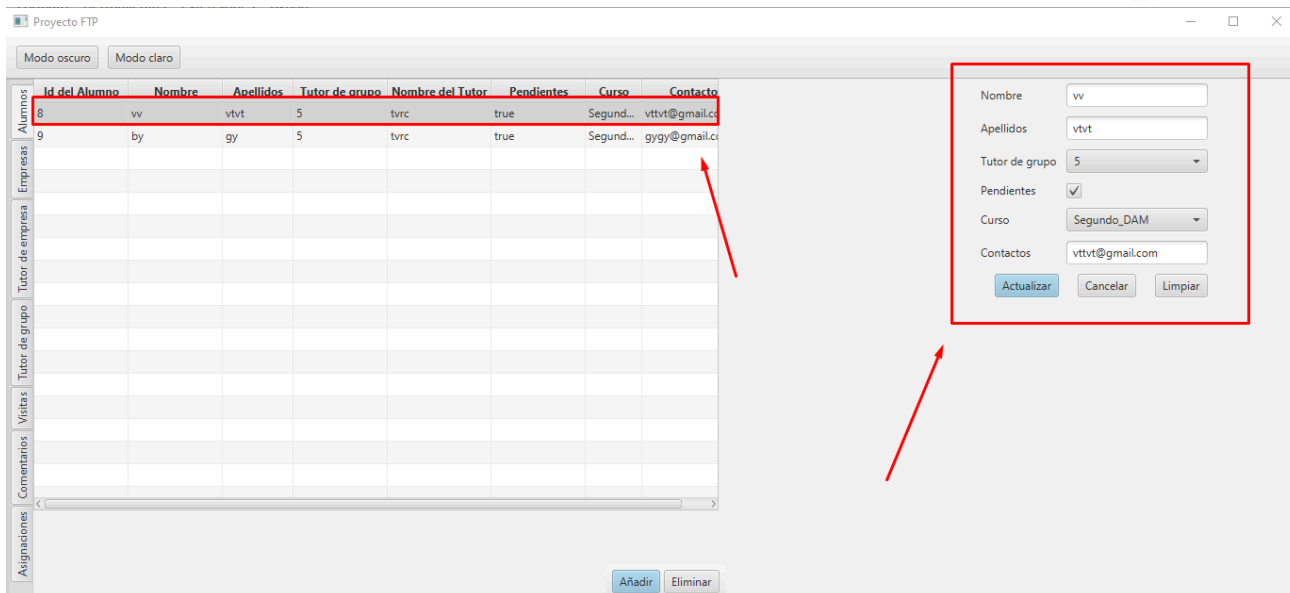
Especialidad: alguna

Añadir Limpiar Cancelar

Añadir Eliminar

En ella deberán introducir todos los campos y pulsar el botón “Añadir” para introducirlos a la base de datos, si pulsamos “Limpiar” se limpiarán todos los campos y “Cancelar” se cerrará la ventana.

Para editar el alumno simplemente debemos pulsar en él y nos aparecerán sus campos a la derecha:

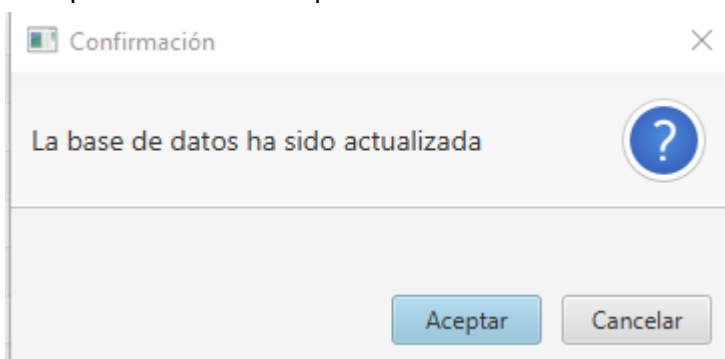


The screenshot shows a web application titled 'Proyecto FTP'. On the left, there is a sidebar with navigation links: 'Alumnos', 'Empresas', 'Tutor de empresa', 'Tutor de grupo', 'Visitas', 'Comentarios', and 'Asignaciones'. The main area displays a table with columns: 'Id del Alumno', 'Nombre', 'Apellidos', 'Tutor de grupo', 'Nombre del Tutor', 'Pendientes', 'Curso', and 'Contacto'. Two rows are visible, with the first row highlighted in red. To the right of the table, a form is open for editing the selected student. The form contains fields for 'Nombre', 'Apellidos', 'Tutor de grupo', 'Pendientes', 'Curso', and 'Contactos', each with a corresponding input field or dropdown menu. At the bottom of the form are three buttons: 'Actualizar', 'Cancelar', and 'Limpiar'. Red arrows indicate the flow from the table row to the form and from the 'Actualizar' button back to the table.

Id del Alumno	Nombre	Apellidos	Tutor de grupo	Nombre del Tutor	Pendientes	Curso	Contacto
8	vv	vtvt	5	tvrc	true	Segund...	vttvt@gmail.co
9	by	gy	5	tvrc	true	Segund...	gygy@gmail.co

Cambiamos los campos y presionamos el botón “Actualizar”.

Nos aparecerá una alerta que nos informará que la base de datos se ha actualizado.



The screenshot shows a confirmation dialog box titled 'Confirmación'. It contains the text 'La base de datos ha sido actualizada' and a question mark icon. At the bottom, there are two buttons: 'Aceptar' and 'Cancelar'.

Si seleccionamos un alumno y pulsamos el botón “Eliminar”.

[illegible]

Aparecerá una alerta si deseamos eliminar el Alumno seleccionado:

Si pulsamos “Aceptar” se eliminará el alumno, sin embargo pulsamos “Cancelar” no se eliminará y nos mostrará la tabla de los alumnos

4.1: Código de la interfaz

En este apartado veremos cómo funciona la aplicación a nivel interno. Como todas las clases controlador son casi iguales solo tomaremos una como referencia, en este caso veremos los controladores relacionados al manejo de la tabla tutor de grupo.

Antes de empezar veremos la parte importante del RootController, que es el controlador padre, desde el que los demás se visualizarán.

```
@Override
public void initialize(URL location, ResourceBundle resources) {

}

public RootController(){ 2 usages
    try {
        FXMLLoader fxmlLoader = new FXMLLoader(getClass().getResource( name: "/fxml/RootView.fxml"));
        fxmlLoader.setController(this);
        fxmlLoader.load();
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
}
```

En la imagen anterior podemos ver como el campo RootController accede al fxml que contiene la vista padre con las pestañas.

Ahora sí, vamos a ver las clases que manejan la tabla tutor grupo.

Empecemos por el evento que controla el botón añadir.

```
47      @FXML no usages
48      void onAñadirAction(ActionEvent event) {
49          tutor_grupoCreateController tutor_grupoCreateController = new tutor_grupoCreateController();
50          Stage stage = new Stage();
51          stage.setTitle("Nuevo tutor de grupo");
52          stage.setScene(new Scene(tutor_grupoCreateController.getRoot()));
53          stage.showAndWait();
54          if (tutor_grupoCreateController.isConfirmar()){
55              tutor_grupoController tutor_grupoController = new tutor_grupoController();
56              String nombre = tutor_grupoCreateController.getNombre().getText();
57              String grupo = tutor_grupoCreateController.getGrupoCBox().getValue().toString();
58              int nuevoId = insertarTutorGrupo(nombre, grupo);
59              if (nuevoId > 0) {
60                  cargarTablaTutorGrupo();
61              } else {
62                  Alert alerta = new Alert(Alert.AlertType.ERROR);
63                  alerta.setTitle("Error");
64                  alerta.setHeaderText("No se pudo añadir el tutor.");
65                  alerta.setContentText("Hubo un problema al insertar el tutor en la base de datos.");
66                  alerta.showAndWait();
67              }
68          }
69      }
```

En esta imagen podemos ver como el evento llama al controlador de tutor_grupoCreateController y crea la stage con su título y su escena, además mostrar esta etapa y espera a que se oculte (cierre) antes de regresar a la persona que llama. Esto último quiere decir que una vez se abra la ventana de creación no se abrirá una nueva hasta que se cierre la primera, evitando la creación “infinita” de ventanas hasta que se consuma la RAM. Por último, cuando el evento recibe la acción de añadir, en este caso llamada “**isConfirmar**”, recoge la información de los campos e inicializa la inserción en la base de datos mediante el método **insertarTutorGrupo**, que veremos más adelante, y carga el contenido de la base de datos en la tabla. En caso de que se produjera algún error manda una alerta y cancela la inserción.

A continuación veremos el método para insertar en la base de datos.

```
private int insertarTutorGrupo(String nombre, String grupo){ 1 usage
    String sql = "INSERT INTO tutor_grupo (Nombre,Grupo) VALUES (?,?)";
    try (Connection connection = Conectar.getConnection();
        PreparedStatement stmt = connection.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS)) {
        stmt.setString( parameterIndex: 1, nombre);
        stmt.setString( parameterIndex: 2, grupo);
        int rowsAffected = stmt.executeUpdate();
        if (rowsAffected > 0) {
            try (ResultSet generatedKeys = stmt.getGeneratedKeys()) {
                if (generatedKeys.next()) {
                    return generatedKeys.getInt( columnIndex: 1);
                }
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return -1;
}
```

Este método realiza la inserción y la actualización de filas afectadas en la base de datos, estableciendo la conexión e insertando los campos necesarios, a excepción del id que se hace a través del evento anterior. El uso de un PreparedStatement es importante para prevenir inyecciones SQL y permitir la asignación dinámica de parámetros. Se utiliza un catch para el manejo de errores.

Ahora veremos cómo eliminar un campo de la tabla, será de forma similar a la de la inserción.

```
@FXML no usages
void onEliminarAction(ActionEvent event) {
    tutorGrupo selected = tableGrupo.getSelectionModel().getSelectedItem();
    if (selected != null) {
        Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
        alert.setTitle("Eliminar tutor de grupo");
        alert.setHeaderText("Se eliminará el tutor" + selected.getNombre());
        alert.setContentText("¿Estás seguro?");
        Optional<ButtonType> result = alert.showAndWait();
        if (result.isPresent() && result.get() == ButtonType.OK) {
            eliminarTutorGrupo(selected.getId_tutor());
            tableGrupo.getItems().remove(selected);
        }
    }
}
```

El código obtiene el elemento seleccionado de la tabla. La variable “**selected**” es una instancia de la clase `tutorGrupo` (que representa un tutor de grupo). Si no hay ningún elemento seleccionado, “**selected**” será null. El “**if**” asegura que solo se proceda con la eliminación si efectivamente hay un elemento seleccionado en la tabla. Si no hay selección, el código dentro del bloque “**if**” no se ejecutará. Si se ha seleccionado un tutor de grupo, se muestra un cuadro de diálogo de confirmación al usuario usando un “**Alert**”. Después de que el usuario elige una opción (OK o Cancelar), el código verifica si el usuario ha seleccionado OK. El método “**result.isPresent()**” verifica si se obtuvo una respuesta, y “**result.get() == ButtonType.OK**” comprueba si esa respuesta es “OK”. Si es así, se procede con la eliminación.

```
private void eliminarTutorGrupo(int tutorgrupoId) { 1 usage
    String sql = "DELETE FROM tutor_grupo WHERE Id_Tutor = ?";
    try (Connection connection = Conectar.getConnection();
        PreparedStatement stmt = connection.prepareStatement(sql)) {
        stmt.setInt(1, tutorgrupoId);
        int rowsAffected = stmt.executeUpdate();
        if (rowsAffected > 0) {
            System.out.println("Alumno eliminado correctamente.");
        } else {
            System.out.println("No se encontró el alumno.");
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

Este método elimina un tutor de la tabla `tutor_grupo` de la base de datos usando su `Id_Tutor`. Primero establece la conexión a la base de datos, prepara la consulta de eliminación, ejecuta la consulta, y luego verifica si la eliminación fue exitosa. Si hubo algún problema durante el proceso, el error se captura y se imprime.

```
@Override
public void initialize(URL location, ResourceBundle resources) {
    idTutor.setCellValueFactory(new PropertyValueFactory<>("idTutor"));
    Nombre.setCellValueFactory(new PropertyValueFactory<>("nombre"));
    Grupo.setCellValueFactory(new PropertyValueFactory<>("grupo"));
    cargarTablaTutorGrupo();
    tableGrupo.setItems(tutorGrupo);
    idTutor.setCellValueFactory((CellDataFeatures<tutorGrupo, Integer> cellData -> cellData.getValue().id_tutorProperty().asObject());
    Nombre.setCellValueFactory((CellDataFeatures<tutorGrupo, String> cellData -> cellData.getValue().nombreProperty());
    Grupo.setCellValueFactory((CellDataFeatures<tutorGrupo, String> cellData -> cellData.getValue().grupoProperty().asString());
    tableGrupo.getSelectionModel().selectedItemProperty().addListener((ObservableValue<extends tutorGrupo> observable, tutorGrupo oldValue, tutorGrupo newValue) -> {
        detallePanel.setVisible(newValue != null);
        if (newValue != null) {
            cargar(newValue);
        }
    });
}
```

Este código configura la `TableView` en `JavaFX` para mostrar la lista de tutores. Las columnas de la tabla están vinculadas a las propiedades de los objetos en la lista. El método también maneja la selección de elementos en la tabla, mostrando un panel con detalles adicionales cuando se selecciona un tutor. Además, los datos se cargan dinámicamente en la tabla y se actualizan si se cambia la selección.

“**`cellData.getValue().id_tutorProperty().asObject()`**” Obtiene la propiedad `id_tutor` del objeto de la fila (que es un tipo `IntegerProperty` en este caso) y la convierte en un objeto. “**`getSelectionModel().selectedItemProperty()`**” Obtiene la propiedad de selección de la tabla. Esta propiedad indica el ítem que está seleccionado en la tabla.

“**`addListener()`**” Se agrega un listener que escucha cualquier cambio en la selección de la tabla. Cuando el usuario selecciona un elemento diferente, este listener se activa.

Este método se encarga de cargar una nueva vista que muestra los detalles del tutor seleccionado en una tabla. Utiliza FXMLLoader para cargar un archivo FXML que contiene la interfaz de usuario (UI) correspondiente, luego pasa el objeto de tutor seleccionado al controlador de esa vista, y finalmente actualiza un panel en la UI principal con la nueva vista cargada. Esto permite mostrar los detalles del tutor en un área específica de la interfaz gráfica.

```
public void cargar(tutorGrupo tutorGrupoSeleccionado) { 1 usage
    try {
        FXMLLoader loader = new FXMLLoader(getClass().getResource( name: "/fxml/tutor_grupoSelected.fxml"));
        Pane pane = loader.load();
        tutor_grupoSelectedController tutor_grupoSelectedController = loader.getController();
        tutor_grupoSelectedController.obtenerTutorGrupo(tutorGrupoSeleccionado);
        detallePanel.getChildren().clear();
        detallePanel.getChildren().add(pane);
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
}
```

El método cargarTablaTutorGrupo se conecta a la base de datos, ejecuta una consulta SQL para obtener todos los registros de la tabla tutor_grupo, crea un objeto tutorGrupo para cada registro y lo agrega a la lista. Luego, esta lista se vincula a la tabla en la interfaz de usuario para mostrar los datos. Esto permite visualizar los tutores en la tabla de manera dinámica y actualizarla con los datos de la base de datos.

Con esto terminamos las clases controller y podemos pasar a las createdController. Estas clases son muy simples así que las veremos rápido.

```
private void cargarTablaTutorGrupo() { 2 usages
    String sql = "SELECT * FROM tutor_grupo";
    try (Connection connection = Conectar.getConnection();
        PreparedStatement stmt = connection.prepareStatement(sql);
        ResultSet rs = stmt.executeQuery()) {

        while (rs.next()) {
            tutorGrupo.add(new tutorGrupo(
                rs.getInt( columnLabel: "Id_Tutor"),
                rs.getString( columnLabel: "Nombre"),
                Curso.valueOf(rs.getString( columnLabel: "grupo"))
            ));
        }
        tableGrupo.setItems(tutorGrupo);
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

Este código maneja acciones del usuario en una interfaz gráfica JavaFX.

onActualizarAction: Marca la acción como confirmada (confirmar = true) y cierra la ventana.

onLimpiarAction: Resetea los campos de texto y opciones seleccionables en la interfaz.

```
@FXML no usages
void onActualizarAction(ActionEvent event) {
    confirmar = true;
    cerrar();
}

@FXML no usages
void onLimpiarAction(ActionEvent event) { limpiar(); }

@FXML no usages
void onCancelarAction(ActionEvent event) { cerrar(); }

private boolean confirmar = false; 3 usages

private void limpiar(){ 1 usage
    Nombre.setText("");
    grupoCBox.getItems().clear();
}

private void cerrar(){ 2 usages
    Stage stage = (Stage) root.getScene().getWindow();
    stage.close();
}
```

onCancelarAction: Cierra la ventana sin confirmar ninguna acción.

Cerrar: Se cierra utilizando el nodo raíz de la escena (root) para obtener el Stage.

Por último nos queda ver los `seletedController` que tiene dos métodos principales.

```
@FXML no usages
void onActualizarAction(ActionEvent event) {
    tutor_grupoController tutor_grupo = new tutor_grupoController();
    String nombre = Nombre.getText();
    String grupo = grupoCBox.getValue() != null ? grupoCBox.getValue().name() : null;
    Integer Id_grupo = Integer.parseInt(tutor_grupo.getIdTutor().getText());
    String sql = "UPDATE tutor_grupo SET Nombre = ?, Grupo = ? WHERE Id_grupo = ?";
    try (Connection con = Conectar.getConnection();
        PreparedStatement stmt = con.prepareStatement(sql)) {
        stmt.setInt(parameterIndex: 0, Id_grupo);
        stmt.setString(parameterIndex: 1, nombre);
        stmt.setString(parameterIndex: 2, grupo);
        int rowsUpdated = stmt.executeUpdate();
        if (rowsUpdated > 0) {
            System.out.println("El tutor ha sido actualizado con éxito.");
            Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
            alert.setHeaderText("Base de datos actualizada");
        } else {
            System.err.println("No se encontró el tutor con ese ID.");
        }
    } catch (Exception e) {
        System.err.println("Error al actualizar el tutor: " + e.getMessage());
    }
}
```

Este método, `onActualizarAction`, tiene como objetivo actualizar los datos de un tutor en la base de datos cuando se activa. Toma los valores de los campos de entrada, actualiza los datos de un tutor en la base de datos mediante una consulta SQL y proporciona retroalimentación visual y en la consola sobre el resultado.

El método `obtenerTutorGrupo` tiene como propósito vincular un objeto de tipo `tutorGrupo` con los componentes de la interfaz gráfica para permitir la edición de sus datos. Establece una sincronización entre el modelo (`tutorGrupo`) y los componentes de la interfaz gráfica (`Nombre` y `grupoCBox`). Se incluyen capturas de excepciones para manejar casos en los que los valores del modelo no coincidan con los valores permitidos en la enumeración `Curso`, asegurando que la aplicación no falle y proporcionando información sobre el error.

```
public void obtenerTutorGrupo(tutorGrupo grupo) { 1 usage
    if (grupo != null) {
        Nombre.textProperty().bindBidirectional(grupo.nombreProperty());
        grupoCBox.setItems(FXCollections.observableArrayList(dad.Model.Curso.values()));
        try {
            grupoCBox.getSelectionModel().select(dad.Model.Curso.valueOf(String.valueOf(grupo.getGrupo())));
        } catch (IllegalArgumentException e) {
            System.err.println("Valor no válido para Curso: " + grupo.getGrupo());
        }
        grupoCBox.getSelectionModel().selectedItemProperty().addListener((ObservableValue<extends Curso> observable, Curso oldValue, Curso newValue) -> {
            if (newValue != null) {
                grupo.setGrupo(Curso.valueOf(newValue.name()));
            }
        });
        grupo.grupoProperty().addListener((ObservableValue<extends Curso> observable, Curso oldValue, Curso newValue) -> {
            if (newValue != null) {
                try {
                    grupoCBox.getSelectionModel().select(dad.Model.Curso.valueOf(String.valueOf(newValue)));
                } catch (IllegalArgumentException e) {
                    System.err.println("Valor no válido para Curso: " + newValue);
                }
            }
        });
    }
}
```

En algunas clases podemos encontrar estas funciones:

```
private void validarCampo(TextField textField, String regex, String errorMessage) { 3 usages
    textField.textProperty().addListener((ObservableValue<extends String> observable, String oldValue, String newValue) -> {
        if (!newValue.matches(regex)) {
            textField.setStyle("-fx-border-color: red; -fx-border-width: 2px;");
            Tooltip tooltip = new Tooltip(errorMessage);
            Tooltip.install(textField, tooltip);
        } else {
            textField.setStyle(null);
            Tooltip.uninstall(textField, tooltip);
        }
    });
}
```

Con esto podemos controlar que ingresa el usuario en los “TextFields” ya que si por ejemplo el usuario ingrese un número el “TextField” se pondrá en rojo hasta que el usuario lo introduzca correctamente

Para saber cuando el usuario ha introducido un campo correctamente tenemos esta función:

```
private boolean validarCampos() { 1 usage
    boolean nombreValido = !Nombre.getText().isBlank() && Nombre.getText().matches( regex: "[a-zA-ZáéíóúÁÉÍÓÚñÑ ]+");
    boolean apellidosValidos = !Apellidos.getText().isBlank() && Apellidos.getText().matches( regex: "[a-zA-ZáéíóúÁÉÍÓÚñÑ ]+");
    boolean contactoValido = (
        !Contactos.getText().isBlank() && Contactos.getText().matches( regex: "^[\\+]?[0-9]{1,4}?[\\s.-]?\\((? [0-9]{1,4}\\)?)?[\\s.-]?[0-9]{1,4}[\\s.-]?[0-9]{1,9}");
        (!Contactos.getText().isBlank() && Contactos.getText().matches( regex: "[\\w.%+-]+@[\\w.-]+\\.([a-zA-Z]{2,})$"));
    return nombreValido && apellidosValidos && contactoValido;
}
```

Con esto cuando el usuario introduzca un valor erróneo ya se dentro de Nombre, Apellidos o Contacto, la función devolverá un false y aparecerá el TextField correspondiente en rojo, hasta que el usuario lo rellene correctamente y desaparecerá el color rojo del TextField.

Y para ejecutarlo añadimos estas líneas en el inicializable de la clase:

```
@Override
public void initialize(URL location, ResourceBundle resources) {
    Curso.setItems(FXCollections.observableArrayList(dad.Model.Curso.values()));
    validarCampo(Nombre, regex: "[a-zA-ZáéíóúÁÉÍÓÚñÑ ]+)", errorMessage: "Introduzca el Nombre correctamente.");
    validarCampo(Apellidos, regex: "[a-zA-ZáéíóúÁÉÍÓÚñÑ ]+)", errorMessage: "Introduzca los apellidos correctamente.");
    validarCampo(Contactos, regex: "^[\\+]?[0-9]{1,4}?[\\s.-]?\\((? [0-9]{1,4}\\)?)?[\\s.-]?[0-9]{1,4}[\\s.-]?[0-9]{1,9}";
    cargarIdAlumno();
}
```

5: Conexión HikariCP

Para asegurarnos de una conexión fluida y ligera entre la base de datos y nuestra aplicación hemos implementado un pool de conexiones utilizando HikariCP, un componente reconocido por su eficiencia y rendimiento en la gestión de conexiones con bases de datos.

A continuación se explicará la forma en la que se ha implementado esta herramienta.

La configuración inicial del pool se realiza a través de la clase HikariConfig, en la que se definen varios parámetros fundamentales:

El siguiente método permite a otras partes del programa obtener una conexión del pool:

```
static {
    try {
        // Configuración inicial conectándose a la base de datos del sistema
        HikariConfig config = new HikariConfig();
        config.setDriverClassName("com.mysql.cj.jdbc.Driver");
        config.setJdbcUrl("jdbc:mysql://localhost:3306/mysql"); // Usamos 'mysql' como conexión inicial
        config.setUsername("root");
        config.setPassword("");
        config.setMaximumPoolSize(10);
        config.setMinimumIdle(2);
        dataSource = new HikariDataSource(config);
    }
}
```

Para verificar que la base de datos que se va a utilizar existe se hace la siguiente comprobación:

```
try (Connection connection = dataSource.getConnection();
     Statement statement = connection.createStatement()) {

    ResultSet resultSet = statement.executeQuery("SHOW DATABASES LIKE 'gestion'");
    if (!resultSet.next()) {
        // Si la base de datos no existe, crearla
        statement.executeUpdate("CREATE DATABASE gestion");
        System.out.println("Base de datos 'gestion' creada.");
    } else {
        System.out.println("La base de datos 'gestion' ya existe.");
    }
}

// Reconfigurar HikariCP para usar la base de datos 'gestion'
config.setJdbcUrl("jdbc:mysql://localhost:3306/gestion");
dataSource = new HikariDataSource(config);
```

Para el control de errores en esta comprobación haremos un catch que mostrara un mensaje con el error producido:

```
} catch (Exception e) {  
    System.err.println("Error configurando la conexión: " + e.getMessage());  
    throw new ExceptionInInitializerError(e);  
}  
}
```

Si hay conexiones libres en el pool, se reutilizan.

Si no hay conexiones libres, pero no se ha alcanzado el límite (MaximumPoolSize), se crea una nueva.

En caso de que todas las conexiones estén ocupadas, el cliente espera hasta que una conexión quede disponible.

```
public static Connection getConnection() throws SQLException {  
    return dataSource.getConnection();  
}
```

Este método cierra todas las conexiones gestionadas por el pool.

```
public static void close() {  
    if (dataSource != null) {  
        dataSource.close();  
    }  
}
```

El código analizado implementa una solución robusta para la gestión de conexiones con bases de datos mediante HikariCP. Este enfoque optimiza el rendimiento y garantiza un manejo eficiente de los recursos.

6. Tecnologías Utilizadas

1.-Lenguaje de Programación:

- Para este caso usaremos Java 22 para desarrollar la aplicación de escritorio.

2.-Frameworks:

- Usaremos JavaFX para la creación de la interfaz gráfica de usuario.

3.- Base de datos:

- MySQL como sistema de gestión de base de datos.

4.-Gestión de Conexión:

- HikariCP para gestionar el pool de conexiones a la base de datos

Bibliografía

-Stackoverflow
-GitHub

Conclusión

El desarrollo del sistema de gestión de prácticas profesionales para la institución educativa ha logrado cumplir con los objetivos establecidos, centralizando y optimizando la gestión de datos relacionados con empresas colaboradoras, alumnos, tutores, asignaciones y seguimiento de prácticas.

El proyecto destaca por la integración eficiente de una base de datos relacional diseñada en MySQL, que soporta un modelo robusto con relaciones claras entre las entidades, garantizando integridad y escalabilidad. Además, la implementación del pool de conexiones HikariCP asegura un rendimiento óptimo en el acceso a la base de datos, incluso bajo alta concurrencia, mejorando la experiencia del usuario.

La interfaz gráfica, desarrollada en JavaFX, ofrece un entorno intuitivo y funcional que facilita las operaciones CRUD, promoviendo un uso sencillo y eficiente por parte de los usuarios. Esta funcionalidad se refuerza con prácticas de desarrollo seguras, como el uso de consultas preparadas para evitar inyecciones SQL.

En cuanto a los beneficios, el sistema proporciona una solución integral para el seguimiento y evaluación de las prácticas, reduciendo tiempos de gestión, mejorando la trazabilidad de los datos y fortaleciendo la comunicación entre los actores involucrados.

En resumen, este proyecto constituye una herramienta moderna, eficiente y adaptable para la gestión de prácticas profesionales, satisfaciendo las necesidades de la institución y estableciendo una base sólida para futuras ampliaciones o mejoras.