Universidad Mariano Gálvez de Guatemala
Facultad de Ingeniería en sistemas de información y ciencias de la computación
Programación I
Ing. Miguel Catalán

MANUAL TÉCNICO DE PROYECTO BIBLIOTECA MARE MENTIUM

Marely Alejandra Camey González
Carné: 7590-21-2326
Sección A
Guatemala 31 de mayo de 2024

Contenido

INTRODUCCION

Este manual técnico está diseñado para proporcionar una guía detallada sobre el desarrollo, configuración y mantenimiento de un proyecto de JavaFX con conexión a una base de datos, utilizando los principios de la Programación Orientada a Objetos (POO). Este proyecto, denominado “Biblioteca Mare Mentium”, es una aplicación de gestión de bibliotecas que permite a los usuarios realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en los registros de libros, gestionar usuarios, y realizar búsquedas avanzadas en la base de datos.

El principal objetivo de este proyecto es desarrollar una aplicación de escritorio eficiente y fácil de usar para la gestión de bibliotecas. La aplicación permite a los administradores de la biblioteca gestionar libros y usuarios de manera efectiva. Utilizando JavaFX para la interfaz gráfica de usuario (GUI), PostgreSQL como sistema de gestión de bases de datos y los principios de la POO, este proyecto busca ofrecer una solución robusta y escalable.

Alcance

Este manual está dirigido a desarrolladores y técnicos que participan en el desarrollo, mantenimiento y mejora de la aplicación “Biblioteca Mare Mentium”. El manual está estructurado en secciones claramente definidas para facilitar la navegación y referencia. Cada sección incluye explicaciones detalladas, ejemplos de código, diagramas y capturas de pantalla cuando sea necesario para ilustrar conceptos y procedimientos. Con este manual Técnico se espera proporcionar un recurso completo y útil para cualquier desarrollador o técnico que trabaje en el proyecto “Biblioteca Mare Mentium”, permitiendo una compresión profunda de su arquitectura y facilitando su mantenimiento y expansión futura.

DIAGRAMA DE CASOS DE USO

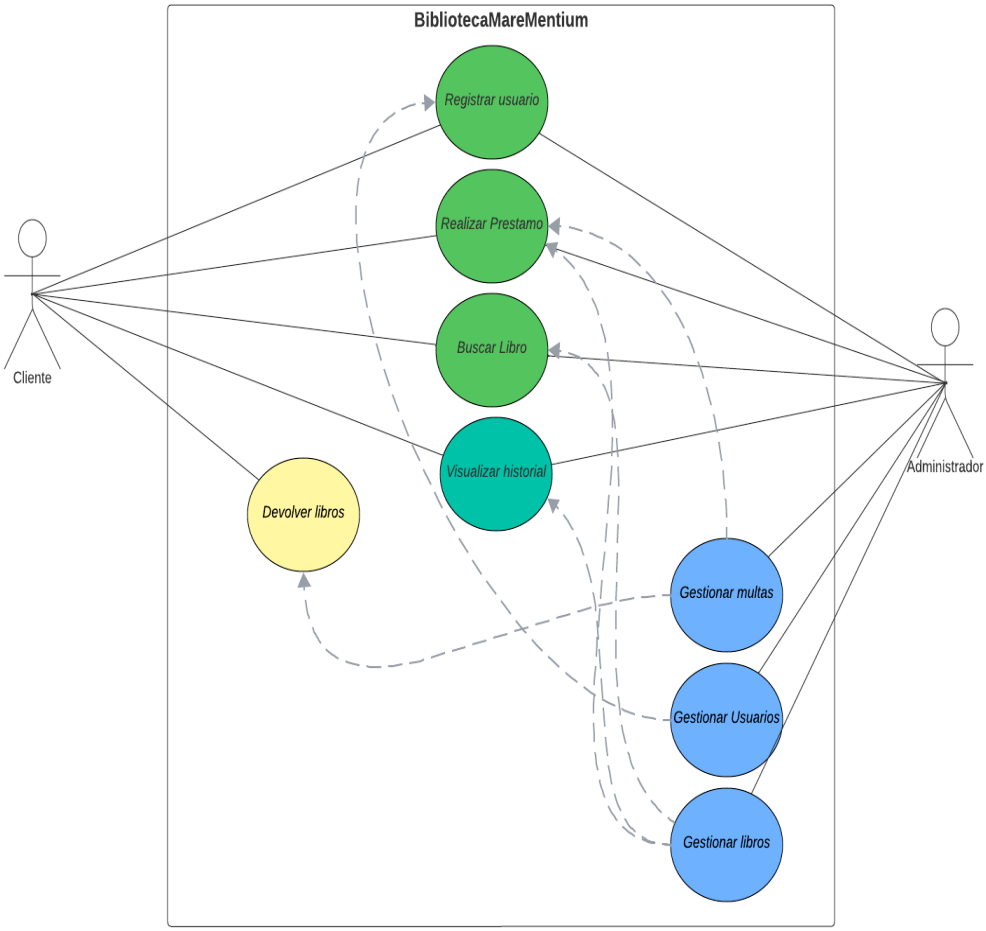


DIAGRAMA DE CLASES

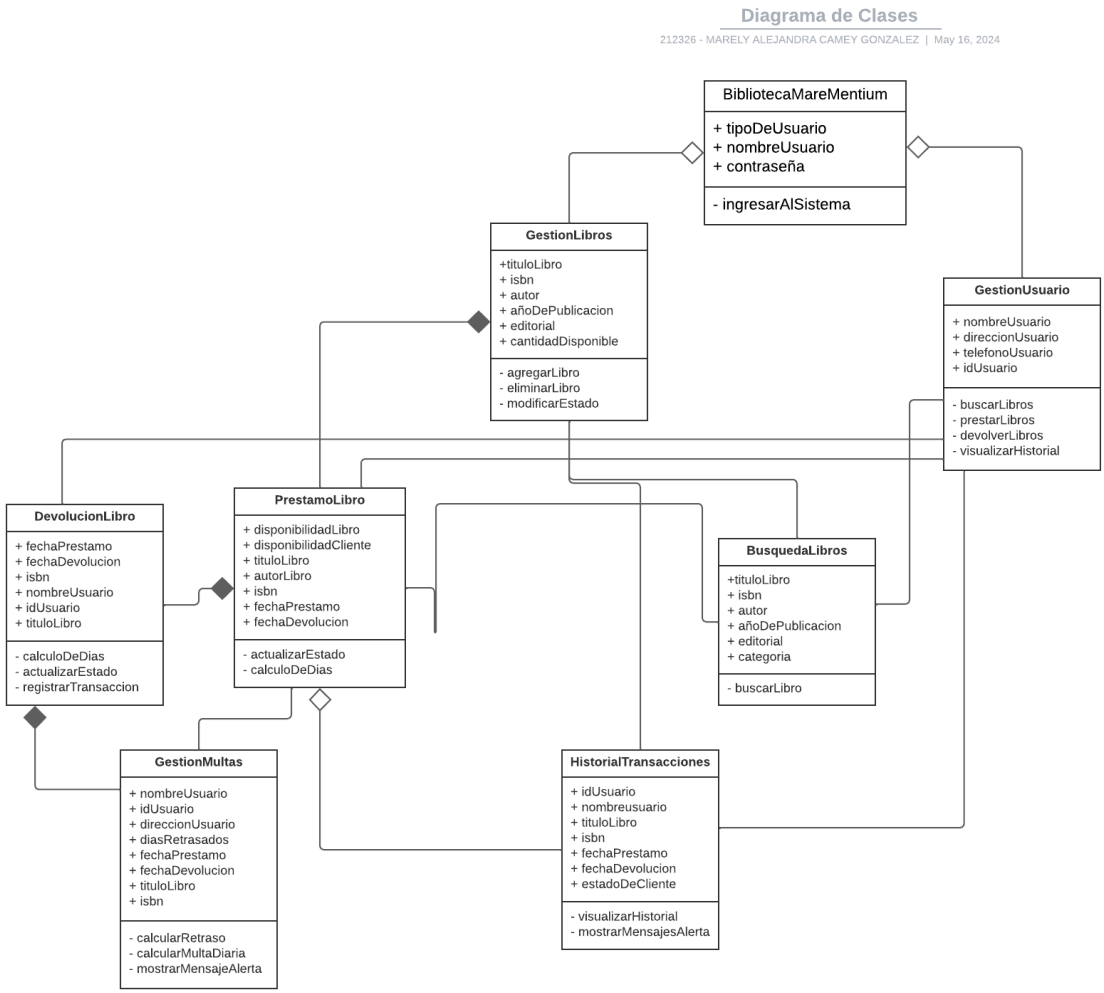
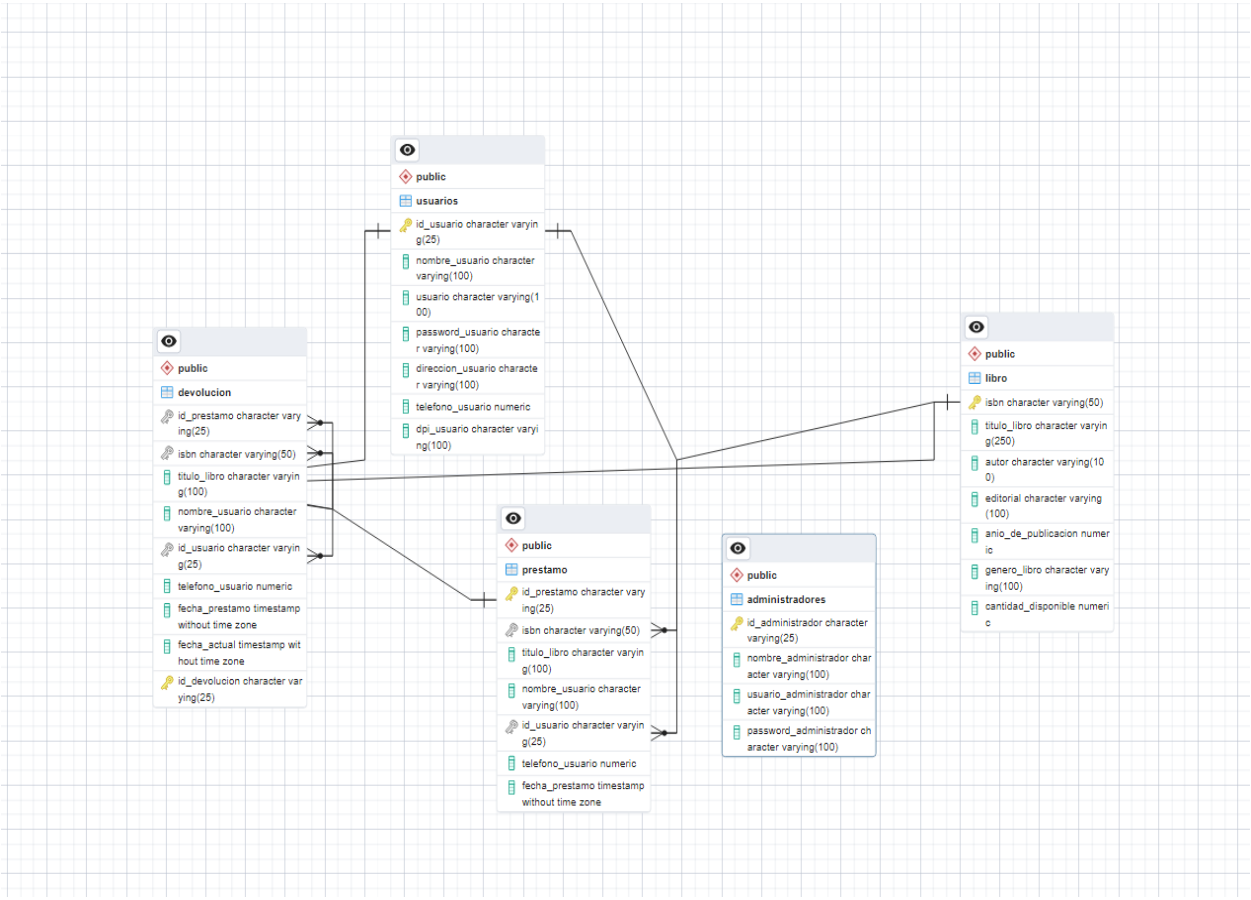


DIAGRAMA DE BASES DE DATOS



TECNOLOGIAS UTILIZADAS

1. Java: Lenguaje de programación principal utilizado para desarrollar la lógica de la aplicación.
2. JavaFX: Biblioteca gráfica utilizada para crear la interfaz del usuario.
3. PostgreSQL: Sistema de gestión de bases de datos relacional utilizado para almacenar y gestionar los datos.
4. JDBC (Java Database Connectivity): API utilizada para conectar la aplicación Java con la base de datos PostgreSQL.
5. NeatBeans: Entorno de desarrollo integrado (IDE) utilizado para el desarrollo del proyecto.
6. Git y GitHub: Herramientas utilizadas para el control de versiones y la colaboración en el desarrollo del código fuente.

INSTRUCCIONES DE CONFIGURACION DEL ENTRONO DE TRABAJO

Instalación y configuración de entorno de Desarrollo:

1. Java Development kit (JDK) 17 o superior:

- Asegurarse de tener el JDK 17 o una versión superior instalada en tu sistema. Puedes descargarlo desde https://www.oracle.com/java/technologies/javase-downloads.html o https://jdk.java.net/.

2. NeatBeans IDE 17

- Descargar e instalar NetBeans IDE desde https://netbeans.apache.org/download/nb17/nb17.html.

3. JafaFX SDK

- Descarga el JavaFX SDK desde https://gluonhq.com/, descargue la versión adecuada para el JavaFX SDK para su sistema operativo.

4. Instalar NetBeans 17

- Descargar el instalador desde el sitio oficial e instalar siguiendo las instrucciones del instalador.

5. Configurar JavaFX en Neatbeans

- Abra NeatBeans y seleccione la opción de herramientas, luego Bibliotecas.
- Cree una nueva biblioteca “JavaFX”
- Añada los archivos JAR del JavaFX SDK descargado, estos se escuentran en la carpeta ‘lib’ del JavaFX SDK

6. Configurar el proyecto

- Abra el proyecto descargado.
- Haga clic derecho en el proyecto y seleccione Propiedades
- Diríjase a Bibliotecas y añada la biblioteca JavaFX creada con anterioridad.

7. Descargar PostgreSQL

- Dirigirse al sitio oficial: https://www.postgresql.org/download/
- Seleccione el sistema operativo de su maquina y siga las instrucciones para descargar el instalador adecuado.

8. Descargar el controlador JDBC

- Dirigirse al sitio oficial de PostgreSQL https://jdbc.postgresql.org/download.html
- Descargue el archivo ‘postgresql-<vresion>.jar’

9. Añadir el JAR a tu Proyecto en NeatBeans

- Abra NeatBeans y el proyecto JavaFX
- Haga clic derecho en el proyecto y seleccione ‘Propiedades’
- Vaya a ‘Bibliotecas’ > ‘Añadir JAR/carpeta ‘
- Navegue hasta donde descargó el archivo y añádalo.

10. Descargue el archivo zip y descomprimalo:
    https://github.com/MACG21/ProgramacionI.git

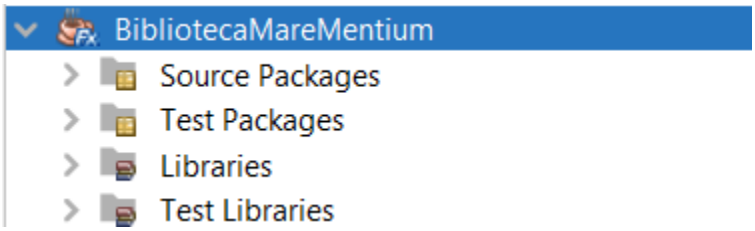
- En el encontrará el proyecto que puede ejecutar en NeatBeans y toda la documentación extra perteneciente al proyecto.

Documentación extra

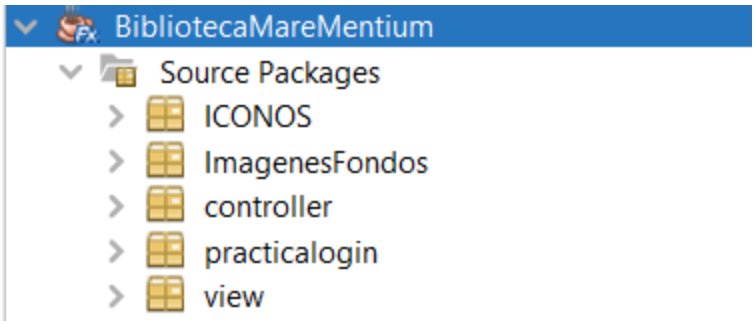
- Documentación de JavaFX: https://openjfx.io/
- NeatBeans Documentation: https://netbeans.apache.org/kb/
- En la descarga del JDK 17. Asegúrese de configurar las variables de entorno JAVA_HOME y PATH.

ESTRUCTURA DEL PROYECTO

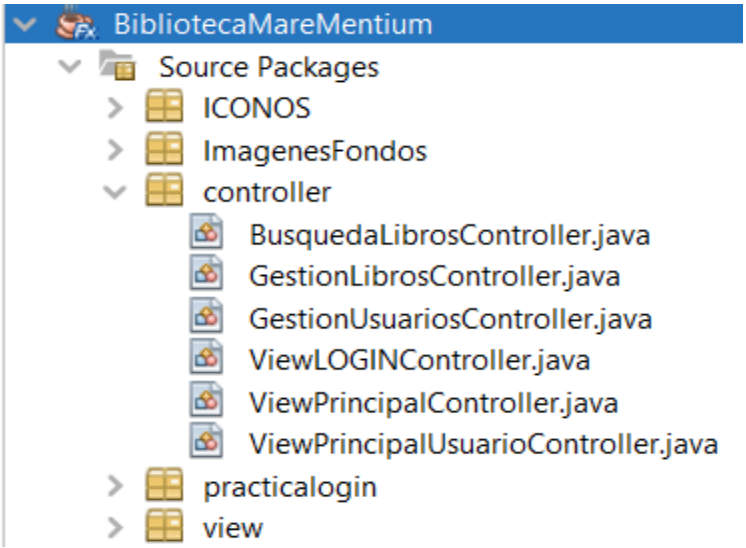
En la estructura principal encontramos los siguientes folders que contienen, el código, los paquetes y librerías necesarias para la ejecución del proyecto.



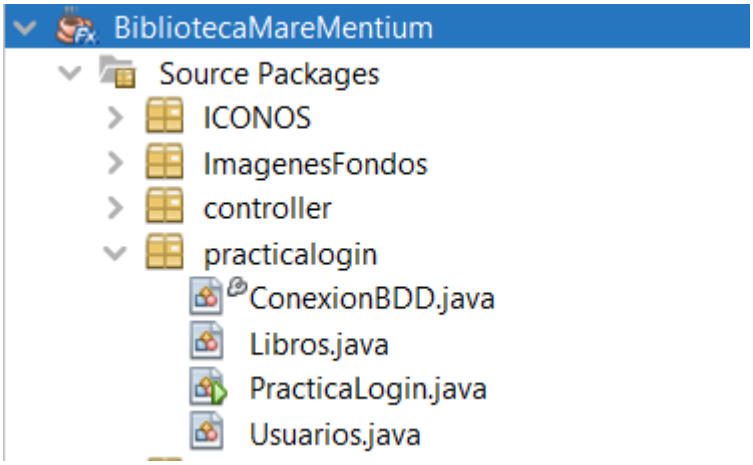
Si desplegamos el folder Source Packages encontraremos los paquetes de cógido y de imágenes e iconos que se visualizan en la interfaz gráfica del proyecto.



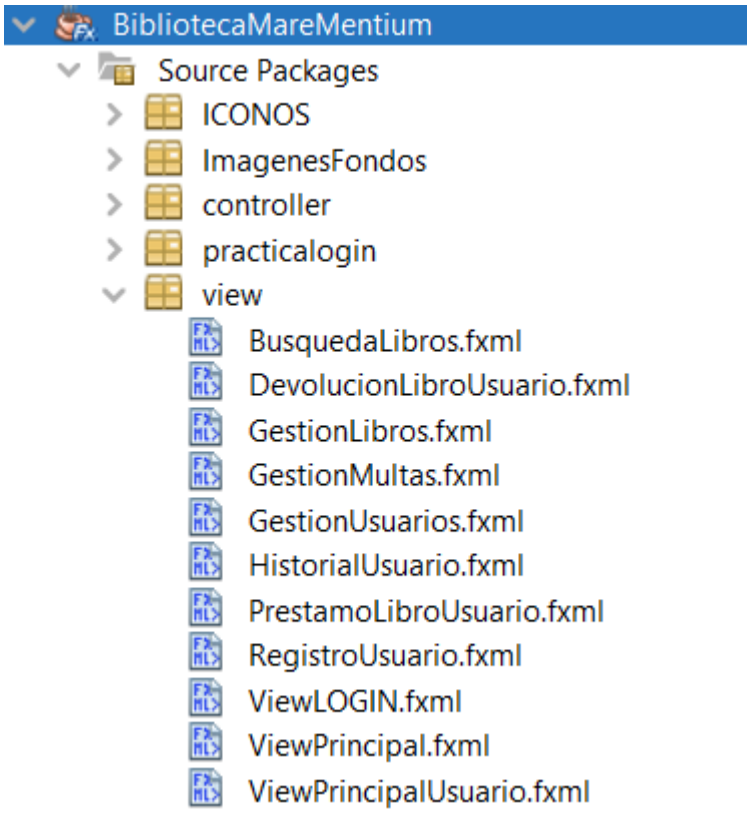
También observamos dentro del package “controller” todos los controller creados de las interfaces gráficas que presenta el proyecto.



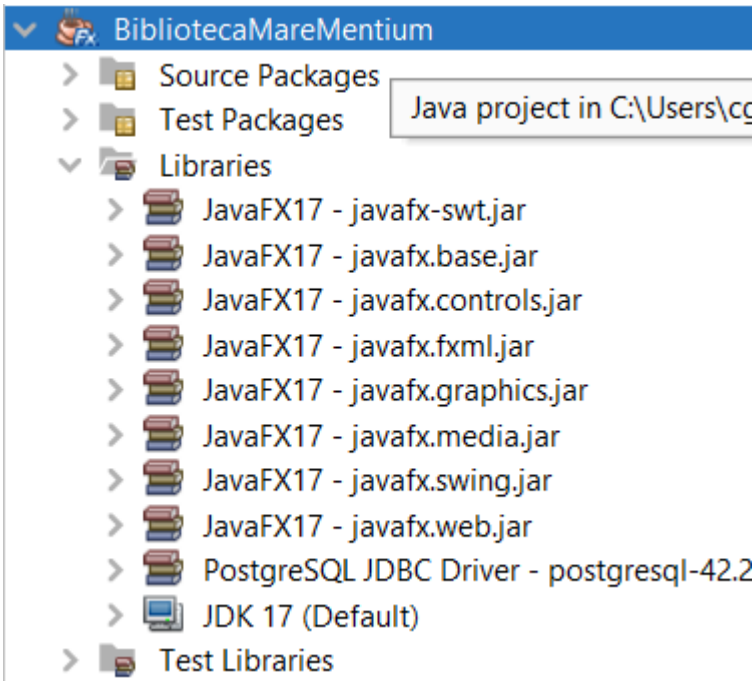
En el siguiente package llamado “practicaLogin” encontramos las clases .java que nos permitirán hacer uso de la conexión a la base de datos y a la abstracción de los objetos. De igual forma tenemos la clase principal “PracticaLogin.java” que es la que contiene el método **main** el cual inicializa el proyecto.



Luego en package llamado “view” se encuentran todos los archivos .fxml que nos permite editar la interfaz gráfica del proyecto.



En el folder de Libraries, se encuentran todas las librerías instaladas para la ejecución del programa.



Interfaz Gráfica de Usuario (GUI):

INICIO DE SESION:

Es la interfaz inicial en la que puedes iniciar sesión con tu usuario y contraseña. Encaso de no poseer cuenta puedes registrarte.

REGISTRO DE USUARIO

1. En caso de que no tengas usuario y presiones el botón registrarte, se te mostrará en pantalla la interfaz que te permitirá agregar tu usuario al sistema, una vez presiones el botón REGISTRAR te dirigirá directamente a la vista principal perteneciente a tu tipo de usuario.

2. Una vez hayas iniciado sesión dependiendo del tipo de usuario que tengas podrás observar la siguiente interfaz de acuerdo con tus necesidades y permisiones:

PERFIL ADMINISTRADOR:

- El perfil del administrador cuenta con cuatro opciones para gestionar las diferentes necesidades de la biblioteca. Este perfil puede gestionar los Usuarios que estén dentro de la base de datos del sistema, al igual que puede gestionar los libros y la búsqueda de estos.

- Al seleccionar el botón de “Gestión de Usuarios”, se muestra una nueva interfaz que nos permite visualizar en una tabla todos los usuarios ya logueados en el sistema, podemos ver los datos de los usuarios como lo son su Nombre, la contraseña (con el propósito de si en alguna emergencia se necesitase la modificación de la misma, solamente los administradores pueden modificar contraseñas), el nombre de usuario, su dirección, teléfono y dpi (aun cuando se piensa que los usuarios de la biblioteca son de origen nacional el campo “DPI” también está pensado para cualquier tipo de modelo de identificación personal, así que se pueden agregar letras, números y espacios en el mismo)

- Puedes seleccionar cualquier registro de usuario y los datos te aparecerán en los campos de texto, dentro de estos mismos puedes hacer la modificación que desees para actualizar la tabla dentro de la base de datos, si quieres hacer más de una modificación por campos, deberás limpiar y seleccionar de nuevo el registro para que el programa no te falle.

  Registro seleccionado:

Luego de modificar el dato que necesitaba ser cambiado, seleccionamos el botón ACTUALIZAR.

Una vez seleccionado, podemos darnos cuenta como el registro se modifica en la tabla.

Al oprimir el botón LIMPIAR los campos de texto quedan vacíos, entonces podemos ingresar los datos nuevos de un usuario si es que deseamos ingresarlo desde acá y el usuario tiene problemas con registrarse solo.

El registro se presenta de manera inmediata en la tabla de usuarios existentes.

En caso de querer eliminar un usuario, podemos seleccionar el registro deseado y presionar el botón ELIMINAR

De esa forma podemos observar que la tabla ya no tiene el registro que acabamos de eliminar.

- Por ultimo el botón de VOLVER nos devuelve a la página inicial del usuario administrador. Donde podemos seguir observando las demás opciones de gestión que podemos escoger.

GESTION DE LIBROS: Dentro de la gestión de libros se nos muestra una interfaz en la que podemos observar los campos de texto que nos permitirán agregar y visualizar textos ya sea para añadir un nuevo registro de libro o para modificar algún registro ya existente.

- Esta interfaz funciona de la misma forma que la gestión de usuarios, la selección dentro de la tabla nos permitirá visualizar en los campos los datos, gracias a esto podemos modificar estos datos ya existentes al oprimir el botón ACTUALIZAR.

- En caso querer agregar un nuevo registro de libro y los campos de texto estén ocupados, podemos seleccionar el botón LIMPIAR que nos ayudará a dejarlos vacíos para el nuevo registro deseado.
- Con el botón AGREGAR podemos agregar el nuevo registro.
- Con el botón ELIMINAR podemos eliminar el registro que hayamos seleccionado con anterioridad.

- Por último, el botón de VOLVER nos devuelve a la página inicial del usuario administrador. Donde podemos seguir observando las demás opciones de gestión que podemos escoger.

BUSQUEDA DE LIBROS: dentro de la búsqueda de libros se nos muestra interfaz en el que podemos observar que nos piden un parámetro, dentro de este parámetro y el campo de texto podemos ingresar el TITULO o el ISBN del libro que deseemos encontrar dentro de la base de datos.

- El registro por buscar se muestra en la tabla una vez se haya escrito el elemento a buscar en el campo de texto y presionado el botón de buscar.

- El botón volver nos regresa a la vista principal del administrador.

Conexión a la Base de Datos:

```
 * @author cgale
 */
public class ConexionBDD {
    // Instancia única de la clase
    private static ConexionBDD instancia;

    // Objeto Connection para la conexión a la base de datos
    private Connection conexion;

    // URL de la base de datos PostgreSQL
    private final String url = "jdbc:postgresql://kala.db.elephantsql.com:5432/rhgiozcx";

    // Propiedades para la conexión (usuario y ocntraseña)
    private Properties properties = new Properties();

    // Objeto estático de Connection
    private static java.sql.Connection conn = null;

    // Construcctor privado para evitar la instanciación desde fuera de la clase
    private ConexionBDD () {
        // Establecer propiedades de usuario y contraseña
        properties.setProperty( key: "user",  value: "rhgiozcx");
        properties.setProperty( key: "password",  value: "PXFhw7JZXY-0CY9afgumUAFWq08fWHEU");

        try {
            // Intentar establecer la conexión a la base de datos
            conn = DriverManager.getConnection(url,  info: properties);
        } catch (SQLException ex) {
            // Manejar cualquier excepción que ocurra durante la conexión
            Logger.getLogger( name: ConexionBDD.class.getName()).log( level: Level.SEVERE,  msg: null,  thrown: ex);
        }
    }
```

```
    //Método estático para obtener la conexión
    public static java.sql.Connection getConnection() {
        // Si la conexión no ha sido creada, crear una nueva instancia de ConexionBDD
        if (conn == null) {
                ConexionBDD c = new ConexionBDD();
                return c.conn;
        }
        else {
            // Si ya existe una conexión, devolverla
            return conn ;
        }
    }
}
```

Propósito del código: proporcionar una forma centralizada y controlada de gestionar la conexión a una base de datos PostgreSQL utilizando el patrón Singleton (Instancia única de la clase), Esto asegura que solo haya una única conexión a la base de datos en cualquier momento, lo cual es útil para:

- Evitar múltiples conexiones innecesarias: al tener una única conexión compartida, se reduce la carga en la base de datos.
- Gestionar recursos de manera eficiente: facilita la gestión de recursos de conexión y asegura un uso más eficiente de las conexiones disponibles.

Implementación de la POO:

PRINCIPIOS DE ABSTRACCION Y ENCAPSULAMIENTO

```
 * @author cgale
 */
public class Libros {

    //Coneccion a BASE DE DATOS
    Connection conn;
    public Libros() {
        conn = ConexionBDD.getConnection();
    }

    //DECLARACION DE VARIABLES Y APLICACION DEL PRINCIPIO DE ABSTRACCION
    // APLICACION DEL PRINCIPIO DE ENCAPSULAMIENTO
    private String isbn;
    private String titulo_libro;
    private String autor;
    private String editorial;
    private int anio_de_publicacion;
    private String genero_libro;
    private int cantidad_disponible;


    //CONSTRUCTOR DE LOS ATRIBUTOS DE LA CLASE
    public Libros(String isbn, String titulo_libro, String autor, String editorial, int anio_de_public
        this.isbn = isbn;
        this.titulo_libro = titulo_libro;
        this.autor = autor;
        this.editorial = editorial;
        this.anio_de_publicacion = anio_de_publicacion;
        this.genero_libro = genero_libro;
        this.cantidad_disponible = cantidad_disponible;
    }

    //GETTERS DE LOS ATRIBUTOS QUE NOS PERMITEN ACCEDER A ELLOS PARA IMPLEMENTARLOS EN OTRAS CLASES
    public String getIsbn() {...3 lines }

    public String getTitulo_libro() {
```

```java
 * @author cgale
 */
public class Usuarios {

    //Conexion a base de dtos
    Connection conn;
        public Usuarios(){
        conn = (Connection) ConexionBDD.getConnection();
    }

    //APLICACION DEL PRINCIPIO DE ABSTRACCION DE POO
        // APLICACION DEL PRINCIPIO DE ENCAPSULAMIENTO
    private String id_usuario;
    private String nombre_usuario;
    private String usuario;
    private String password_usuario;
    private String direccion_usuario;
    private Integer telefono_usuario;
    private String dpi_usuario;


    //CONSTRUCTOR DE LOS ATRIBUTOS DE LA CLASE
    public Usuarios(String id_usuario, String nombre_usuario, String usuario, String password_usuario,
        this.id_usuario = id_usuario;
        this.nombre_usuario = nombre_usuario;
        this.usuario = usuario;
        this.password_usuario = password_usuario;
        this.direccion_usuario = direccion_usuario;
        this.telefono_usuario = telefono_usuario;
        this.dpi_usuario = dpi_usuario;
    }

    //GETTERS DE LOS ATRIBUTOS QUE NOS PERMITEN ACCEDER A ELLOS PARA IMPLEMENTARLOS EN OTRAS CLASES
    public String getId_usuario() {
        return id_usuario;
    }
```

POLIMORFISMO:

```java
    /* METODO QUE NOS PERMITE MOLDEAR EL COMPORTAMIENTO DE LOS BOTONES
                EN LA INTERFAZ PRINCIPAL DEL ADMINISTRADOR*/
    @FXML
    private void eventAction(ActionEvent event) {
        try {
            String fxmlFile = "";
            if (event.getSource() == btnGestionDeLibros) {
                fxmlFile = "/view/GestionLibros.fxml";
            } else if (event.getSource() == btnGestionDeUsuario) {
                fxmlFile = "/view/GestionUsuarios.fxml";
            } else if (event.getSource() == btnCerrarSesion) {
                fxmlFile = "/view/ViewLOGIN.fxml";
            } else if (event.getSource() == btnBusquedaLibros) {
                fxmlFile = "/view/BusquedaLibros.fxml";
            }

            if (!fxmlFile.isEmpty()) {
                changeScene(event, fxmlFile);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

```java
/*METODO QUE NOS PERMITE MOLDEAR EL COMPORTAMIENTO DE LOS BOTONES
        EN EL CONTROLLER DE LA INTERFAZ DE GESTION DE LIBROS*/
@FXML
    private void handleButtonAction(ActionEvent handleButtonAction) throws SQLException, IOException {
    //Implementacion del botón agregar, haciendo uso del metodo agregar registro que hace la inserción en la Base de datos
    if(handleButtonAction.getSource() == btnAgregar){
        agregarRegistro();
    }
    //Implementacion del boton volver, devolviendo a la vista principal del usuario administrador
    else if(handleButtonAction.getSource()== btnVolver){
        Parent root = FXMLLoader.load( url: getClass().getResource( name: "/view/ViewPrincipal.fxml"));
        Stage stage = (Stage) ((Node) handleButtonAction.getSource()).getScene().getWindow();
        Scene scene = new Scene( parent: root);
        stage.setScene(scene);
        stage.show();
        }
    //implementacion del metodo limpiar con el boton limpiar
    else if(handleButtonAction.getSource() == btnLimpiar){
        limpiarTextFiel();
    }
    //impleentacion del metodo actualizar con el boton actualizar
    else if (handleButtonAction.getSource() == btnActualizar){
    actualizar();
    }
    //implementacion del metodo eliminar con el boton elimanar
    else if (handleButtonAction.getSource() == btnEliminar){
    eliminarRegistro();
    }
}
```

```java
/*METODO QUE NOS PERMITE MOLDEAR EL COMPORTAMIENTO DE LOS BOTONES
        EN EL CONTROLLER DE LA INTERFAZ DE GESTION DE USUARIOS*/
@FXML
private void handleButtonAction(ActionEvent event) throws SQLException, IOException {
    //IMPLEMENTACION DEL BOTON AGREGAR, HACIENDO USO DEL METODO CORRESPONDIENTE A ÉL
    if(event.getSource() == btnAgregar){
        agregarUsuario();
    }
    //IMPLEMENTACION DEL BOTON ELIMINAR, HACIENDO USO DEL METODO CORRESPONDIENTE
    else if(event.getSource() == btnEliminar){
        eliminarUsuario();
    }
    //IMPLEMENTACION DEL BOTON ACTUALIZAR, HACINDO USO DEL METODO CORRESPONDIENTE
    else if(event.getSource() == btnActualizar){
        actualizar();
    }
    else if(event.getSource() == btnLimpiar){
        limpiarTextFiel();
    }
    else if(event.getSource() == btnVolver){
    Parent root = FXMLLoader.load( url: getClass().getResource( name: "/view/ViewPrincipal.fxml"));
    Stage stage = (Stage) ((Node) event.getSource()).getScene().getWindow();
        Scene scene = new Scene( parent: root);
        stage.setScene(scene);
        stage.show();
    }
}
```

```java
/*METODO QUE NOS PERMITE MOLDEAR EL COMPORTAMIENTO DE LOS BOTONES
        EN EL CONTROLLER DE LA INTERFAZ DE BUSQUEDA DE LIBROS*/
@FXML
private void handleButtonAction(ActionEvent event) throws IOException {
    // IMPLEMENTACION DEL BOTON BUSCAR
    if(event.getSource() == btnBuscar){
    buscarLibro();
    }
    // IMPLEMENTACION DEL BOTON VOLVER
    else if(event.getSource() == btnVolver){
        Parent root = FXMLLoader.load( url: getClass().getResource( name: "/view/ViewPrincipal.fxml"));
        Stage stage = (Stage) ((Node) event.getSource()).getScene().getWindow();
        Scene scene = new Scene( parent: root);
        stage.setScene(scene);
        stage.show();
    }

}
```

Realmente el principio de abstracción se da en todos los métodos que tengan el mismo nombre, pero se comporten de diferente forma según el tipo de necesidad.

Módulos Funcionales:

Todos los métodos siguen la misma estructura en su implementación en otras clases.

METODO AGREGAR:

```
//METODO QUE NOS PERMITE HACER LA INSERCION DE NUEVOS USUARIOS DESDE EL ADMINISTRADOR
public void agregarUsuario(){
Vector<Object> datosUsuario = new Vector<>();

//AGREGAR LOS DATOS AL VECTOR
datosUsuario.add( e: txtIdUsuario.getText());
datosUsuario.add( e: txtNombre.getText());
datosUsuario.add( e: txtUsuario.getText());
datosUsuario.add( e: txtContraseña.getText());
datosUsuario.add( e: txtDireccion.getText());
datosUsuario.add( e: Integer.valueOf( s: txtTelefono.getText()));
datosUsuario.add( e: txtDpi.getText());

String query = "INSERT INTO Usuarios(Id_Usuario, Nombre_Usuario, Usuario, Password_Usuario, "
        + "Direccion_Usuario, Telefono_Usuario, DPI_Usuario) VALUES (?, ?, ?, ?, ?, ?, ?)";

try(java.sql.Connection conn = ConexionBDD.getConnection(); PreparedStatement stNuevo = conn.prepareStatement( string: query)){
// ESTABLECER LOS VALORES EN EL PREPAREDSTATEMENT DESDE EL VECTOR
stNuevo.setString( i: 1, (String) datosUsuario.get( index: 0));
stNuevo.setString( i: 2, (String) datosUsuario.get( index: 1));
stNuevo.setString( i: 3, (String) datosUsuario.get( index: 2));
stNuevo.setString( i: 4, (String) datosUsuario.get( index: 3));
stNuevo.setString( i: 5, (String) datosUsuario.get( index: 4));
stNuevo.setInt( i: 6, (Integer) datosUsuario.get( index: 5));
stNuevo.setString( i: 7, (String) datosUsuario.get( index: 6));

stNuevo.executeUpdate();
mostrarUsuarios();
limpiarTextFiel();

}catch(Exception ex){
    ex.printStackTrace();
}
        executeQuery(query);
}
```

METODO ELIMINAR:

```
//METODO PARA ELIMINAR REGISTROS DE USUARIO
private void eliminarUsuario() throws SQLException{
String query = "DELETE FROM Usuarios WHERE id_usuario = '" + txtIdUsuario.getText() + "'";
executeQuery(query);
mostrarUsuarios();
}
```

METODO ACTUALIZAR:

```
//METODO QUE PERMITE ACTUALIZAR LOS REGISTROS
private void actualizar() throws SQLException{
    String query = "UPDATE Usuarios SET nombre_usuario = '" + txtNombre.getText() + "', usuario = '" + txtUsuario.getText()
            + "', password_usuario = '" + txtContraseña.getText() + "', direccion_usuario = '" + txtDireccion.getText()
            + "', telefono_usuario = " + txtTelefono.getText() + " WHERE id_usuario = '" + txtIdUsuario.getText() + "'";

    executeQuery(query);
    mostrarUsuarios();
}
```

METODO MOSTRAR:

```
    //METODO QUE PERMITE LA VISUALIZACON DE LOS USUARIOS EN TABLA
    public void mostrarUsuarios() throws SQLException{
    conn = ConexionBDD.getConnection();

    ObservableList<Usuarios> lista = getUsuariosList();

    //ASIGNACION A CADA COLUMNA
    colId.setCellValueFactory(new PropertyValueFactory<Usuarios, String>( string: "id_usuario"));
    colNombre.setCellValueFactory(new PropertyValueFactory<Usuarios, String>( string: "nombre_usuario"));
    colUsuario.setCellValueFactory(new PropertyValueFactory<Usuarios, String>( string: "usuario"));
    colContrasenia.setCellValueFactory(new PropertyValueFactory<Usuarios, String>( string: "password_usuario"));
    colDireccion.setCellValueFactory(new PropertyValueFactory<Usuarios, String>( string: "direccion_usuario"));
    colTelefono.setCellValueFactory(new PropertyValueFactory<Usuarios, Integer>( string: "telefono_usuario"));
    colDpi.setCellValueFactory(new PropertyValueFactory<Usuarios, String>( string: "dpi_usuario"));

    tblTablaUsuarios.setItems( ol: lista);
    }
```

Manejo de errores y Depuración:

Puede que al correr el programa este no muestre la interfaz gráfica de inicio de sesión, para solucionar esto puede darle clic derecho al proyecto, seleccionar “Clean and Build”, una vez termine ese proceso puede volver a dar clic derecho en el proyecto y seleccionar la opción “Run File”. De esta forma ya puede iniciar el proyecto.

SECCION DE SOLUCION DE PROBLEMAS COMUNES

Documentación extra

- Documentación de JavaFX: https://openjfx.io/
- NeatBeans Documentation: https://netbeans.apache.org/kb/
- En la descarga del JDK 17. Asegúrese de configurar las variables de entorno JAVA_HOME y PATH.

INFORMACIÓN DE CONTACTO

Email: mcameyg5@gmail.com