Sistema De Gestión De Eventos

Manual Técnico

Introducción

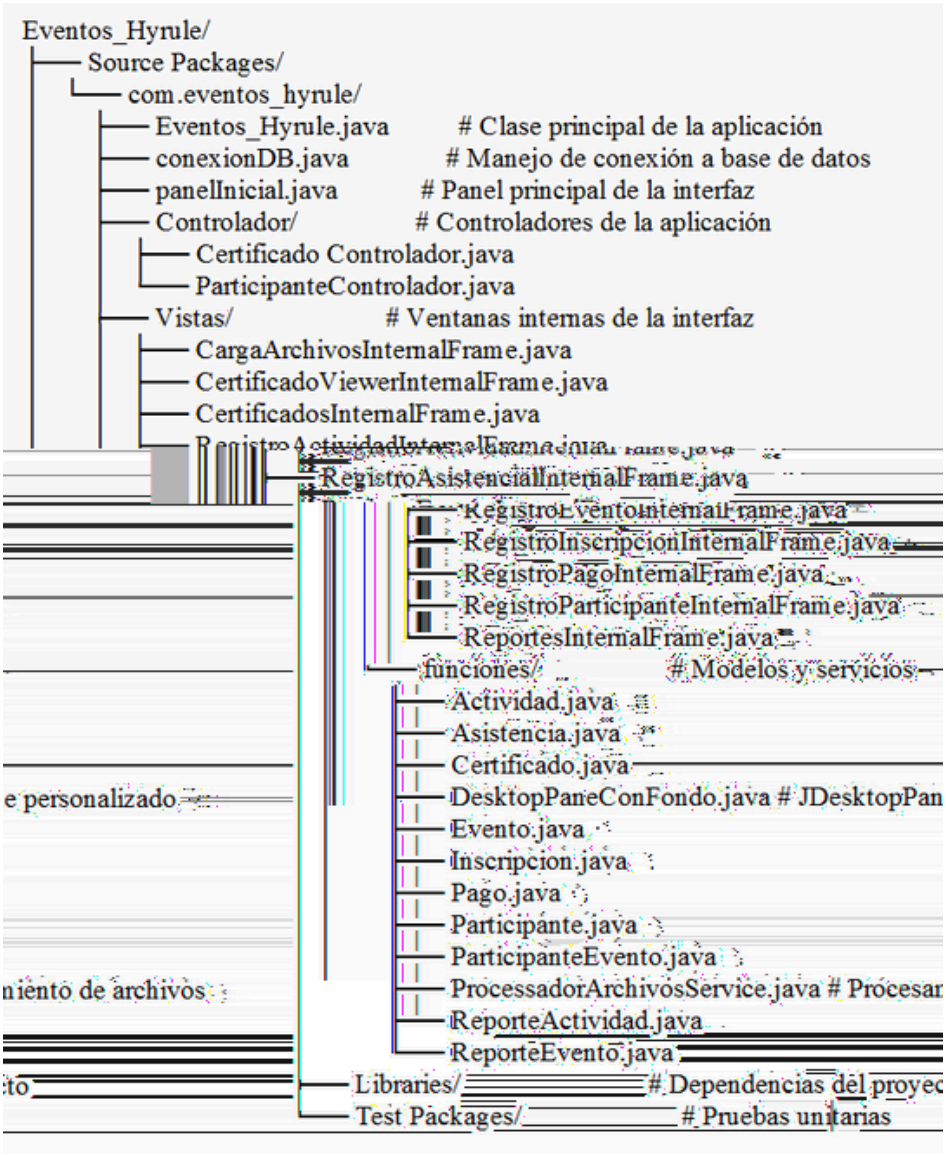
Eventos Hyrule es una aplicación de escritorio desarrollada en Java diseñada para la gestión integral de eventos. El sistema permite el registro de eventos, participantes, actividades, así como el procesamiento de inscripciones, pagos, asistencias y generación de reportes y certificados

Requisitos del sistema:

- ***Sistema Operativo:*** *Windows 10 o superior, Linux o macOS*
- ***JDK:*** *Java 17 o superior*
- ***IDE recomendado:*** *Apache NetBeans IDE 20.*
- ***Base de datos****: MySQL 8.0 o superior*
- ***Control de versiones:*** *Git*
- ***Hardware****:*
  - ***Procesador:*** *1 GHz o superior*
  - ***Memoria RAM:*** *4 GB mínimo*
  - ***Espacio en disco****: 500 MB libres para la instalación*

Estructura del proyecto

- *El proyecto está desarrollado bajo una arquitectura básica de carpetas y archivos.*



Código Fuente:

- *Codigo Fuente: Eventos Hyrule*

Descripción de código

Funcionalidad del Sistema

Procesamiento de archivos de entrada:

- *El sistema procesa archivos de texto con instrucciones estructuradas para realizar operaciones:*
- *Carga del archivo: A través de CargaArchivosInternalFrame.java, el usuario selecciona el archivo, velocidad de procesamiento y ruta de salida*
- *Procesamiento línea por línea: ProcessadorArchivosService.java se encarga de parsear y validar cada línea*
- *Ejecución de instrucciones: Las instrucciones válidas se ejecutan en la base de datos*
- *Registro en log: Todas las operaciones se registran con su resultado*

Instrucciones soportadas:

- *REGISTRO_EVENTO: Registra un nuevo evento*
- *REGISTRO_PARTICIPANTE: Registra un nuevo participante*
- *INSCRIPCION: Inscribe un participante en un evento*
- *PAGO: Registra el pago de una inscripción*
- *VALIDAR_INSCRIPCION: Valida una inscripción después del pago*
- *REGISTRO_ACTIVIDAD: Registra una actividad para un evento*
- *ASISTENCIA: Registra la asistencia a una actividad*
- *CERTIFICADO: Genera un certificado de participación*
- *REPORTE_PARTICIPANTES: Genera reporte de participantes*
- *REPORTE_ACTIVIDADES: Genera reporte de actividades*
- *REPORTE_EVENTOS: Genera reporte de eventos*

Clases Principales:

Eventos_Hyrule.java:

Función: *Clase principal que inicia la aplicación.*

Responsabilidades:

- *Punto de entrada main() de la aplicación*
- *Inicialización de la interfaz gráfica principal*
- *Gestión del ciclo de vida de la aplicación*

conexionDB.java:

Función: *Manejo de conexiones a la base de datos*

Responsabilidades:

- *Establecer y cerrar conexiones con MySQL*
- *Manejar transacciones y control de errores de base de datos*
- *Pool de conexiones para mejor rendimiento*

panelInicial.java:

Función: *Panel principal de la interfaz gráfica*

Responsabilidades:

- *Contener el JDesktopPane principal*
- *Gestionar la apertura y cierre de ventanas internas*
- *Barra de menús principal con todas las opciones*
- *Barra de estado con información del sistema*

Métodos Importantes:

Estos son algunos métodos importantes para comprender la estructura del sistema:

1. ***Inicio y navegación:*** *main(), inicializarUI(), abrirVentanaInterna()*
2. ***Registros:*** *guardarParticipante(), guardarEvento(), realizarInscripcion()*
3. ***Procesamiento:*** *seleccionarArchivo(), procesarArchivo()*
4. ***Pagos:*** *registrarPago(), verificarInscripcion()*
5. ***Reportes:*** *generarReporteEventos(), exportarHTML()*
6. ***Certificados: generarCertificado(), verificarElegibilidad()***
7. ***Búsquedas: buscarPorEmail(), filtrarPorTipo()***

guardarParticipante()

Función: *Insertar depsues de validaciones al i¿participante en la base de datos*

```
private void guardarParticipante() {
    // Validaciones
    if (txtNombre.getText().isEmpty() || txtInstitucion.getText().isEmpty() || txtEmail.getText().isEmpty()) {
        JOptionPane.showMessageDialog(parentComponent: this, message: "Todos los campos son obligatorios", title: "Error", messageType: JOptionPane.ERROR_MESSAGE);
        return;
    }

    // Validar longitud del nombre
    if (txtNombre.getText().length() > 45) {
        JOptionPane.showMessageDialog(parentComponent: this, message: "El nombre no puede exceder 45 caracteres", title: "Error", messageType: JOptionPane.ERROR_MESSAGE);
        return;
    }

    // Validar longitud de institución
    if (txtInstitucion.getText().length() > 150) {
        JOptionPane.showMessageDialog(parentComponent: this, message: "La institución no puede exceder 150 caracteres", title: "Error", messageType: JOptionPane.ERROR_MESSAGE
        return;
    }

    // Validar formato de email
    if (!txtEmail.getText().matches(regex: "^[\\w-\\.]+@([\\w-]+\\.)+[\\w-]{2,4}$")) {
        JOptionPane.showMessageDialog(parentComponent: this, message: "Ingrese un correo electrónico válido", title: "Error", messageType: JOptionPane.ERROR_MESSAGE);
        return;
    }

    // Verificar si el correo ya existe
    if (correoYaRegistrado(email: txtEmail.getText())) {
        JOptionPane.showMessageDialog(parentComponent: this, message: "El correo electrónico ya está registrado", title: "Error", messageType: JOptionPane.ERROR_MESSAGE);
        return;
    }
    // Insertar en la base de datos
    String sql = "INSERT INTO PARTICIPANTE (email_participante, nombre_completo, tipo_participante, institucion) " +
                 "VALUES (?, ?, ?, ?)";
```

procesarArchivos()

Función: *Procesar archivos de entrada*

```
public String procesarArchivo(Path archivoEntrada) throws IOException, SQLException {
    logBuilder.setLength(newLength:0); // Limpiar el log
    agregarLog("Iniciando procesamiento del archivo: " + archivoEntrada.getFileName());
    agregarLog("Directorio de salida: " + directorioSalida);
    agregarLog("Velocidad: " + velocidadMs + " ms por instrucción");
    agregarLog(mensaje:"------------------------------------------------");

    List<String> lineas = Files.readAllLines(path:archivoEntrada);
    int totalLineas = lineas.size();
    int lineasProcesadas = 0;
    int lineasExitosas = 0;

    try (Statement stmt = connection.createStatement()) {
        for (String linea : lineas) {
            lineasProcesadas++;
            if (linea.trim().isEmpty() || linea.trim().startsWith(prefix:"//")) {
                continue; // Saltar líneas vacías o comentarios
            }

            try {
                procesarLinea(linea, numeroLinea:lineasProcesadas);
                lineasExitosas++;
            } catch (SQLException e) {
                agregarLog("   ERROR en línea " + lineasProcesadas + ": " + e.getMessage());
            } catch (Exception e) {
                agregarLog("   ERROR inesperado en línea " + lineasProcesadas + ": " + e.getMessage());
            }

            // Pausa entre instrucciones si es necesario
            if (velocidadMs > 0) {
                try {
                    Thread.sleep(millis:velocidadMs);
                } catch (InterruptedException e) {
                    agregarLog(mensaje:"----- Procesamiento interrumpido -----");
```

cargarEvento()

Función: *Solicitar a la base de datos la informacion de eventos registrados para cargarlos al sistema.*

```
private void cargarEventos() {
    try {
        String sql = "SELECT codigo_evento, titulo_evento FROM EVENTO ORDER BY fecha_evento";
        var stmt = connection.createStatement();
        var rs = stmt.executeQuery(sql);

        cmbEventos.removeAllItems();
        cmbEventos.addItem(item:"-- Seleccione evento --");
        while (rs.next()) {
            cmbEventos.addItem(rs.getString(columnLabel:"codigo_evento") +
                " - " + rs.getString(columnLabel:"titulo_evento"));
        }
    } catch (SQLException e) {
        JOptionPane.showMessageDialog(parentComponent:this, "Error al carga
            title:"Error", messageType:JOptionPane.ERROR_MESSAGE);
    }
}
```

```
ar eventos: " + e.getMessage(),
```

actualizarEstadoInscripcion()

Función: *Manejar el estado de incripción(no inscrito, validada)*

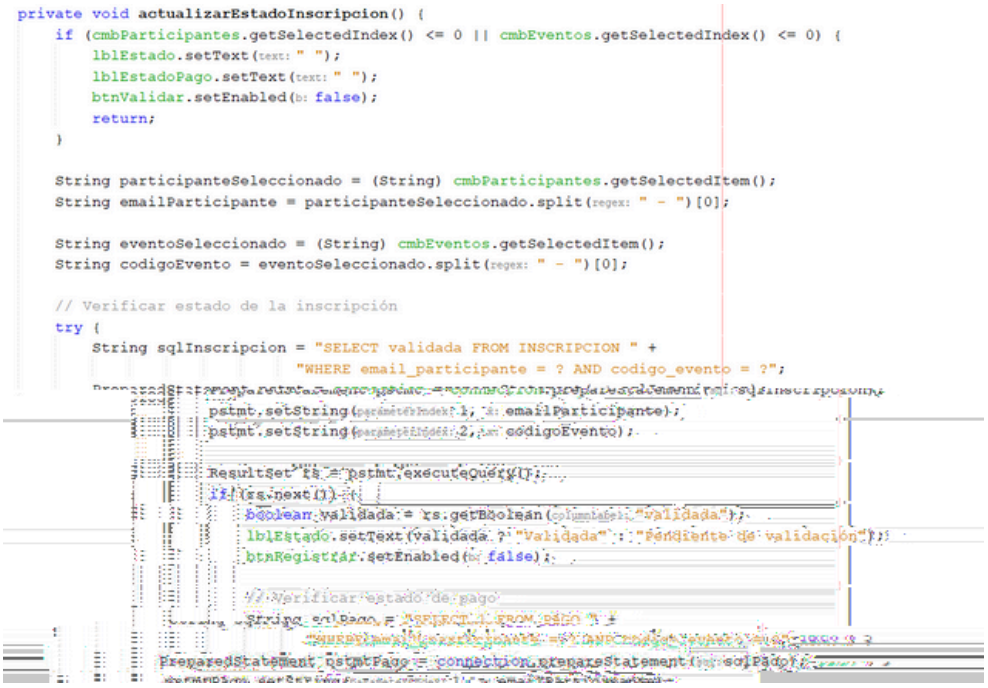


Diagrama Entidad/Relación

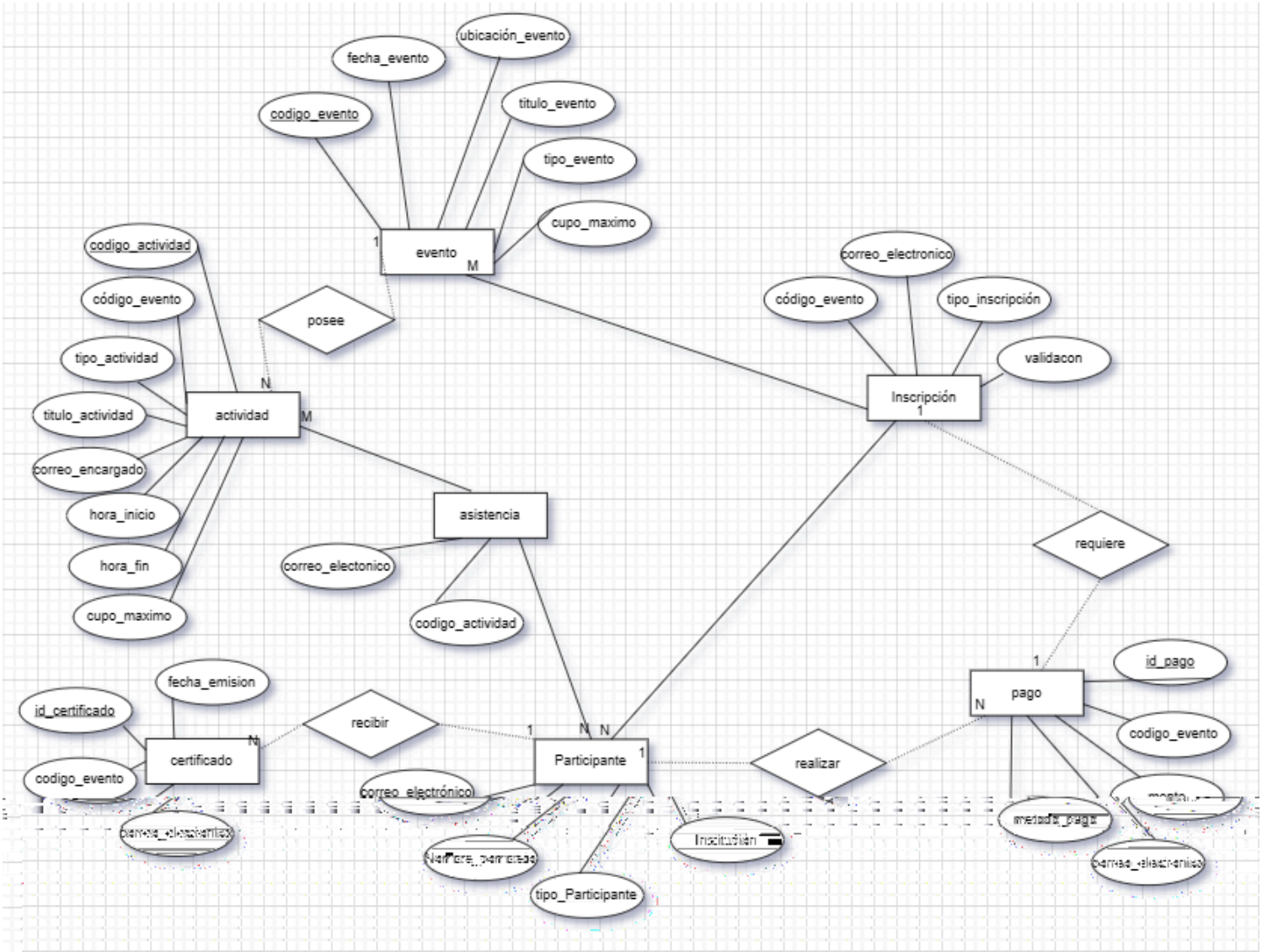
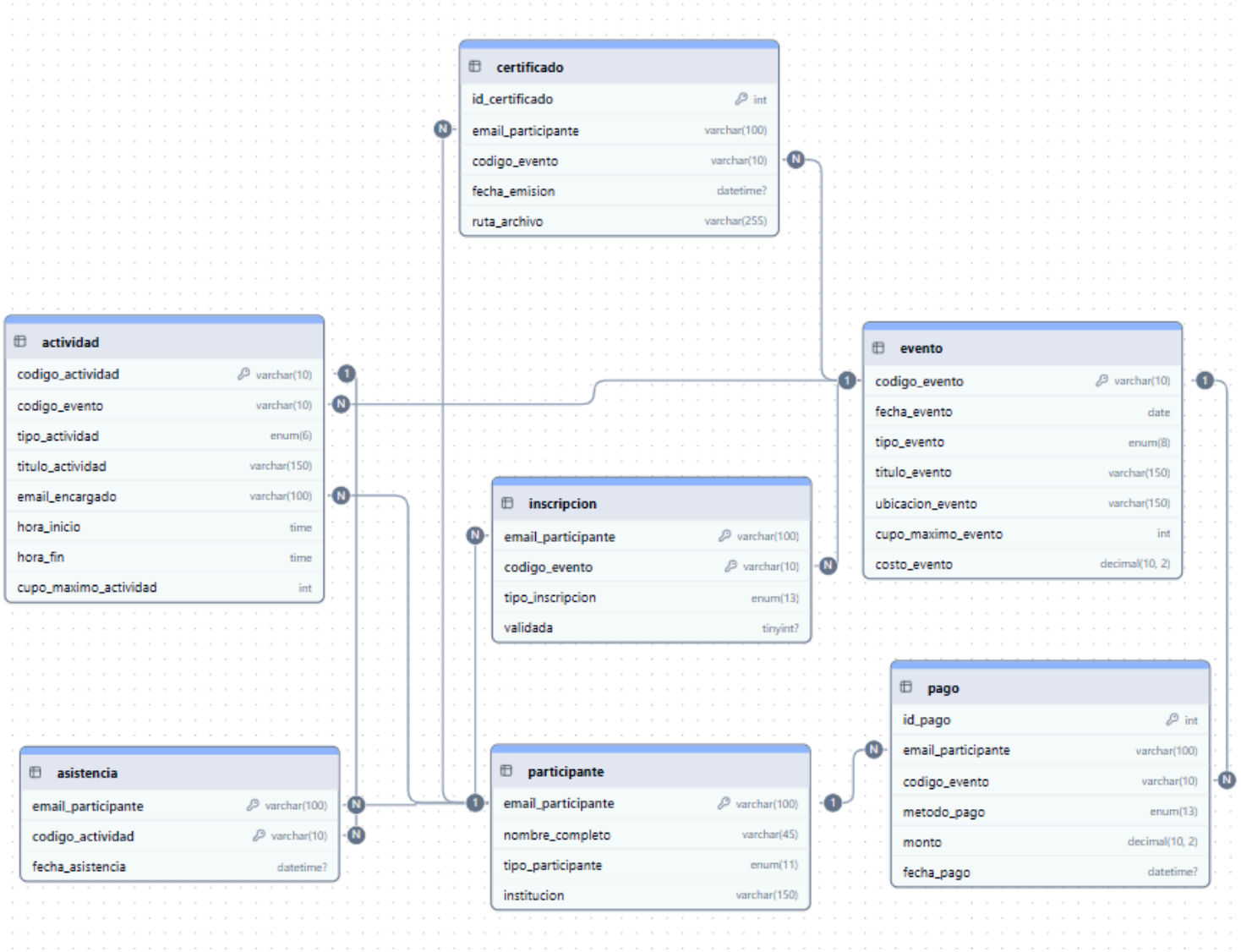


Diagrama de Tablas



Mapeo Fisico de la Base de Datos:

Field	Type	Null	Key	Default	Extra
email_participante	varchar(100)	NO	PRI	NULL	
nombre_completo	varchar(45)	NO		NULL	
tipo_participante	enum('ESTUDIANTE','PROFESIONAL','INVITADO')	NO		NULL	
institucion	varchar(150)	NO		NULL	

Field	Type	Null	Key	Default	Extra
codigo_evento	varchar(20)	NO	PRI	NULL	
fecha_evento	date	NO		NULL	
tipo_evento	enum('CHARLA','CONGRESO','TALLER','DEBATE')	NO		NULL	
titulo_evento	varchar(150)	NO		NULL	
ubicacion_evento	varchar(150)	NO		NULL	
cupo_maximo_evento	int	NO		NULL	
costo_evento	decimal(10,2)	NO		NULL	

Field	Type	Null	Key	Default	Extra
codigo_actividad	varchar(20)	NO	PRI	NULL	
codigo_evento	varchar(20)	YES	MUL	NULL	
tipo_actividad	enum('CHARLA','TALLER','DEBATE','OTRA')	NO		NULL	
tulo_actividad	varchar(150)	NO		NULL	ti
ail_encargado	varchar(100)	NO	MUL	NULL	em
ra_inicio	time	NO		NULL	ho
ra_fin	time	NO		NULL	ho
po_maximo_actividad	int	NO		NULL	cu

Field	Type	Null	Key	Default	Extra
email_participante	varchar(100)	NO	PRI	NULL	
codigo_evento	varchar(20)	NO	PRI	NULL	
tipo_inscripcion	enum('ASISTENTE','CONFERENCISTA','TALLERISTA','OTRO')	NO		NULL	
validada	tinyint(1)	YES		0	

Tabla Pago:

Field	Type	Null	Key	Default	Extra
id_pago	int	NO	PRI	NULL	auto_increment
email_participante	varchar(100)	NO	MUL	NULL	
codigo_evento	varchar(20)	YES	MUL	NULL	
metodo_pago	enum('EFECTIVO','TRANSFERENCIA','TARJETA')	NO		NULL	
monto	decimal(10,2)	NO		NULL	
fecha_pago	datetime	YES		CURRENT_TIMESTAMP	DEFAULT_GENERATED

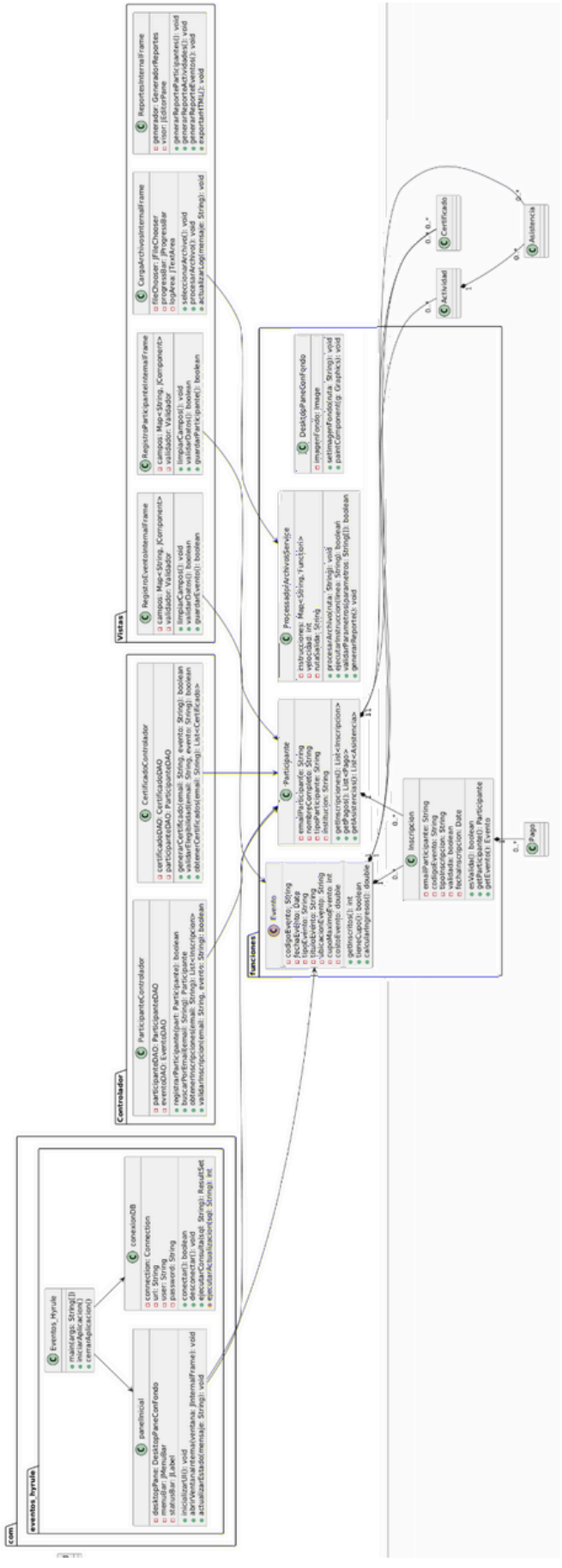
Tabla Asitencia:

Field	Type	Null	Key	Default	Extra
email_participante	varchar(100)	NO	PRI	NULL	
codigo_actividad	varchar(20)	NO	PRI	NULL	
fecha_asistencia	datetime	YES		CURRENT_TIMESTAMP	DEFAULT_GENERATED

Tabla Certificado:

Field	Type	Null	Key	Default	Extra
id_certificado	int	NO	PRI	NULL	auto_increment
email_participante	varchar(100)	NO	MUL	NULL	
codigo_evento	varchar(10)	NO	MUL	NULL	
fecha_emision	datetime	YES		CURRENT_TIMESTAMP	DEFAULT_GENERATED
ruta_archivo	varchar(255)	NO		NULL	

Diagrama de Clases:



Validaciones:

El sistema implementa las siguientes validaciones:

- *Control de cupo en eventos y actividades*
- *Verificación de pago antes de validar inscripciones*
- *Prevención de inscripciones duplicadas*
- *Restricción de certificados a participantes con al menos una asistencia*
- *Validación de formatos de fechas, emails y otros datos*
- *Verificación de integridad referencial en la base de datos*

Flujo de la aplicación

1. *Inicio: La aplicación se inicia desde Eventos_Hyrule.java y se conecta a la base de datos mediante conexionDB.java*
2. *Interfaz principal: Se muestra panelInicial.java con el DesktopPaneConFondo*
3. *Carga de archivo: El usuario puede cargar un archivo de instrucciones mediante CargaArchivosInternalFrame.java*
4. *Procesamiento: Las instrucciones se ejecutan mediante ProcessadorArchivosService.java*
5. *Operaciones manuales: El usuario puede realizar operaciones mediante los formularios correspondientes*
6. *Generación de reportes: Los reportes se generan en HTML en la carpeta especificada*
7. *Visualización de certificados: Los certificados se visualizan en CertificadoViewerInternalFrame.java*

Instalación y Configuración

1. *Instalar Java JDK 17 o superior*
2. *Instalar MySQL 8.0 o superior y crear la base de datos*
3. *Clonar el repositorio Git del proyecto*
4. *Configurar las conexiones a la base de datos en conexionDB.java*
5. *Importar el proyecto en NetBeans o el IDE preferido*
6. *Compilar y ejecutar el proyecto*

Mantenimiento y soporte

Para garantizar el correcto funcionamiento del sistema:

- *Verificar regularmente la conexión a la base de datos en conexionDB.java*
- *Mantener actualizadas las dependencias del proyecto*
- *Realizar copias de seguridad periódicas de la base de datos*
- *Monitorear el espacio en disco para la generación de reportes*

Información del Desarrollador

Desarrollador Principal:
Dayana Sofía Orozco Mendóza
Fecha de Creación: *Agosto 2025*
Tecnologías Utilizadas:

- *Lenguaje de programación: Java*
- *Base de datos: MySQL*
- *Control de versiones: Git*
- *Interfaz gráfica: Java Swing (JDesktopPane y JInternalFrame).*