Universidad de San Carlos de Guatemala

Centro Universitario de Occidente

División de Ciencias de la Ingeniería

Introducción a la Programación y Computación 2

Docente: Ing. Moisés Granados G.

Manual Técnico Practica #1

Estudiante: Juan Fernando García Natareno

RA: 202230077

El presente manual técnico describe el funcionamiento, arquitectura y lineamientos de uso de la aplicación desarrollada por Triforce Software para la administración de eventos en el Reino de Hyrule. El sistema ha sido diseñado con el objetivo de optimizar la gestión integral de eventos, permitiendo a los organizadores llevar un control eficiente sobre el registro de eventos, participantes, actividades, inscripciones, pagos, validaciones y generación de reportes.

La aplicación ofrece dos modos principales de interacción:

- Procesamiento automático mediante archivos de entrada estructurados, que permiten cargar instrucciones para la configuración inicial del sistema.
- Interacción manual a través de una interfaz gráfica amigable, desarrollada con Java Swing y organizada con un JDesktopPane, que facilita la navegación entre múltiples ventanas internas.
- El sistema integra reglas de negocio y validaciones esenciales, tales como:
- Control de cupo en actividades.
- Verificación de pagos antes de validar inscripciones.
- Prevención de inscripciones duplicadas.
- Emisión de certificados únicamente para los participantes que hayan asistido al menos a una actividad validada.

Asimismo, la aplicación cuenta con un módulo de reportes que genera salidas en formato HTML, las cuales se almacenan automáticamente en la carpeta definida por el usuario.

En cuanto a la tecnología utilizada, el sistema ha sido desarrollado en Java, empleando Swing para la interfaz gráfica y MySQL como sistema de gestión de base de datos, garantizando robustez, escalabilidad y facilidad de mantenimiento.

Este manual está dirigido principalmente a desarrolladores y administradores técnicos, proporcionando las directrices necesarias para la instalación, configuración, mantenimiento y extensión del sistema.

Objetivo General:

Desarrollar y documentar una aplicación de escritorio para la administración de eventos en el Reino de Hyrule, que permita gestionar de manera eficiente los procesos de registro, inscripción, actividades, pagos, validaciones y generación de reportes, garantizando la integridad de la información y facilitando su uso mediante una interfaz gráfica intuitiva.

Objetivos Específicos

1. Implementar la carga y procesamiento de archivos de entrada con instrucciones estructuradas que automaticen el registro de eventos, participantes y actividades
2. Diseñar una interfaz gráfica con Java Swing, organizada mediante un JDesktopPane, que facilite la navegación entre formularios y la ejecución manual de las operaciones.
3. Integrar validaciones lógicas y de negocio, como control de cupos, verificación de pagos, prevención de duplicados y control de asistencia para la emisión de certificados.
4. Conectar la aplicación con una base de datos MySQL, asegurando la persistencia, seguridad y consistencia de la información administrada.
5. Generar reportes en formato HTML, exportados automáticamente a la carpeta de salida definida por el usuario.

Datos de MySQL

Mysql –u adminDBA -p

User: adminDBA

Constraseña: admin@123

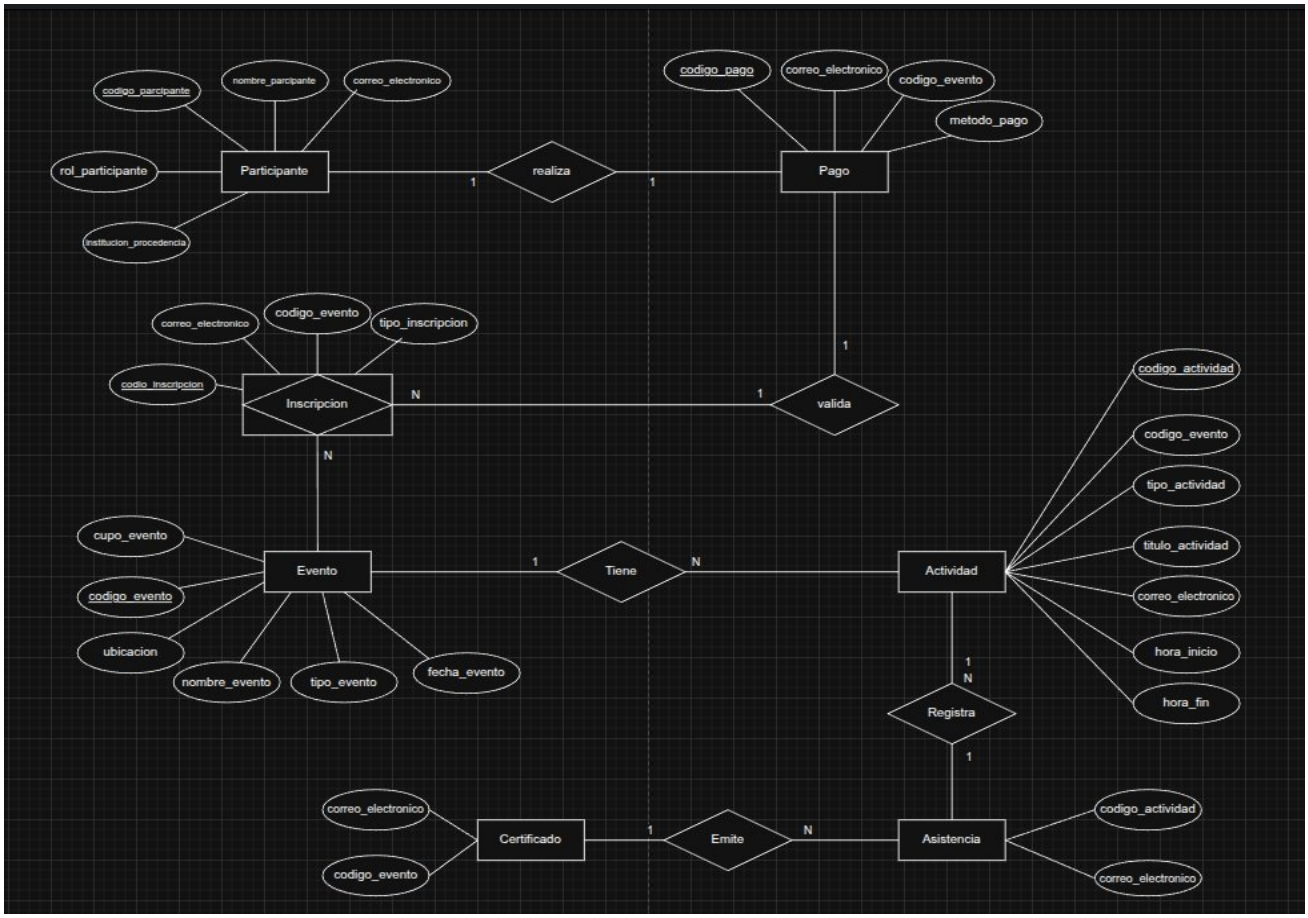
Diagramas E/R y de tablas:

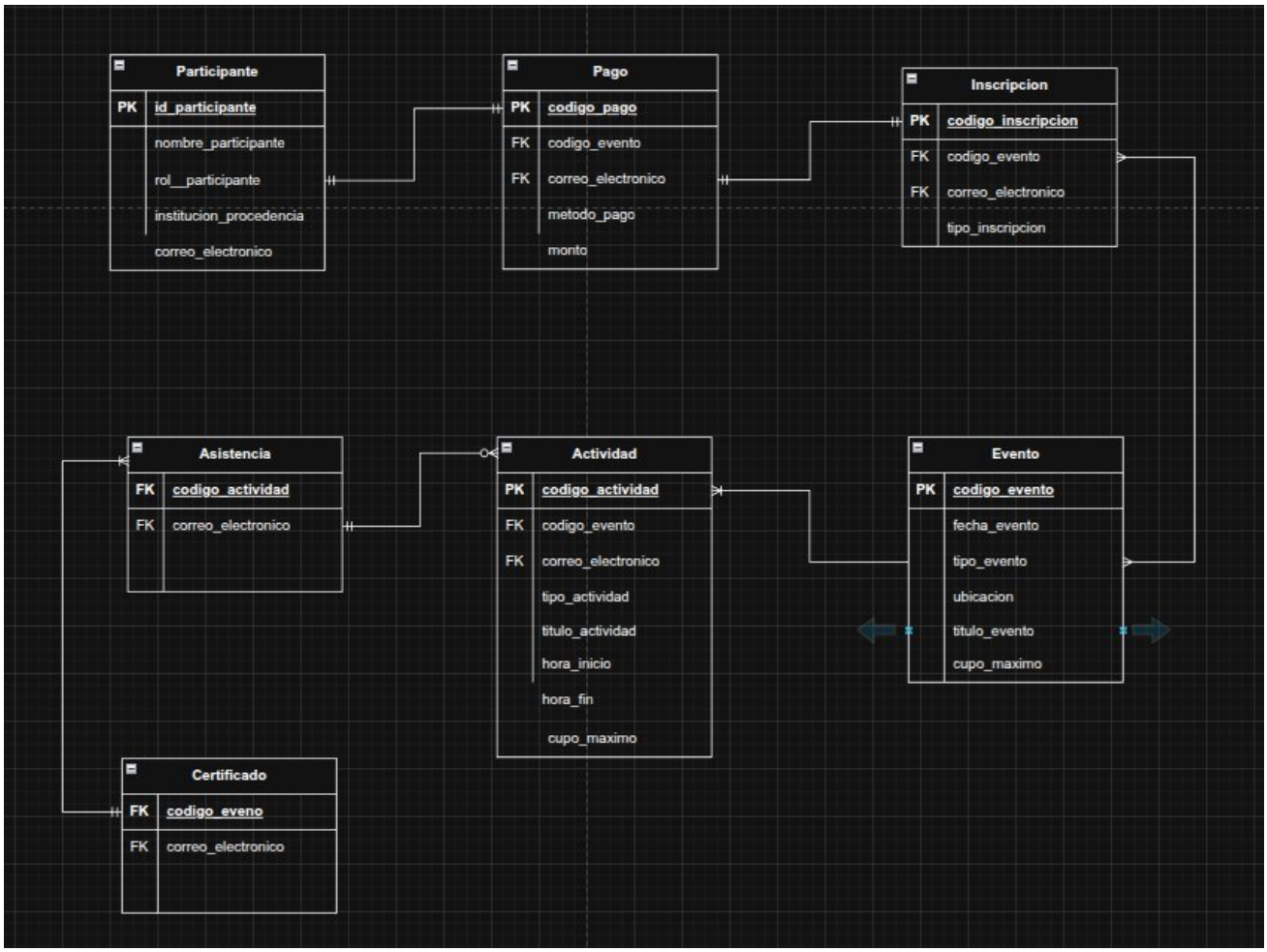
El diagrama muestra las entidades principales del sistema y sus relaciones:

- Participante: Contiene información sobre los participantes, incluyendo codigo_participante, nombre_participante, correo_electronico, rol_participante e institucion_procedencia. Un participante puede realizar un Pago.
- Pago: Registra los pagos realizados por los participantes, con atributos como codigo_pago, correo_electronico, codigo_evento y metodo_pago. Cada pago valida una inscripción.
- Inscripción: Representa la inscripción de un participante a un evento. Se relaciona con Evento y con los pagos que la validan. Incluye atributos como codigo_inscripcion, codigo_evento, correo_electronico y tipo_inscripcion.
- Evento: Define los eventos disponibles en el sistema, con atributos como codigo_evento, nombre_evento, tipo_evento, fecha_evento y cupo_evento. Un evento puede tener múltiples Actividades.
- Actividad: Representa las actividades dentro de un evento, con atributos codigo_actividad, codigo_evento, tipo_actividad, titulo_actividad, correo_electronico, hora_inicio y hora_fin. Cada actividad puede registrar múltiples asistencias.
- Asistencia: Registra la asistencia de participantes a actividades, relacionando codigo_actividad y correo_electronico. Cada asistencia puede generar un Certificado.
- Certificado: Entidad que emite certificados a los participantes que asistieron a las actividades, asociando correo_electronico y codigo_evento.

Relaciones principales:

- Un Participante realiza un Pago (1:1).
- Un Pago valida una Inscripción (1:1).
- Una Inscripción pertenece a un Evento (N:1).
- Un Evento tiene varias Actividades (1:N).
- Una Actividad registra múltiples Asistencias (1:N).
- Una Asistencia puede emitir un Certificado (1:N).


codigo_parcipante
nombre_parcipante
correo_electronico
rol_participante
Participante
institucion_procedencia
realiza
1
1
codigo_pago
correo_electronico
codigo_evento
metodo_pago
Pago
1
correo_electronico
codigo_evento
tipo_inscripcion
codigo_inscripcion
Inscripcion
N
1
valida
N
codigo_actividad
codigo_evento
tipo_actividad
titulo_actividad
correo_electronico
hora_inicio
hora_fin
cupo_evento
codigo_evento
ubicacion
nombre_evento
tipo_evento
fecha_evento
Evento
1
Tiene
N
Actividad
1
N
Registra
1
correo_electronico
codigo_evento
Certificado
1
Emite
N
Asistencia
codigo_actividad
correo_electronico


Participante
PK id_participante
nombre_participante
rol__participante
institucion_procedencia
correo_electronico
Pago
PK codigo_pago
FK codigo_evento
FK correo_electronico
metodo_pago
monto
Inscripcion
PK codigo_inscripcion
FK codigo_evento
FK correo_electronico
tipo_inscripcion
Asistencia
FK codigo_actividad
FK correo_electronico
Actividad
PK codigo_actividad
FK codigo_evento
FK correo_electronico
tipo_actividad
titulo_actividad
hora_inicio
hora_fin
cupo_maximo
Evento
PK codigo_evento
fecha_evento
tipo_evento
ubicacion
titulo_evento
cupo_maximo
Certificado
FK codigo_eveno
FK correo_electronico

Mapeo Físico de la Base de Datos:

```
CREATE TABLE evento(
        codigo_evento VARCHAR(7) NOT NULL,
        fecha_evento DATE NOT NULL,
        tipo_evento ENUM("CHARLA", "CONGRESO", "TALLER", "DEBATE") NOT NULL,
        titulo_evento VARCHAR(25) NOT NULL,
        ubicacion VARCHAR(150) NOT NULL,
        cupo_maximo INT NOT NULL,
        CONSTRAINT PK_codigo_evento PRIMARY KEY (codigo_evento)
);

CREATE TABLE participante (
        nombre_participante VARCHAR(45) NOT NULL,
        rol_participante ENUM("ESTUDIANTE", "PROFESIONAL", "INVITADO") NOT NULL,
        institucion_procedencia VARCHAR(150) NOT NULL,
        correo_electronico VARCHAR(50) NOT NULL,
        CONSTRAINT PK_email PRIMARY KEY (correo_electronico)
);

CREATE TABLE inscripcion (
        codigo_inscripcion INT NOT NULL AUTO_INCREMENT,
        codigo_evento VARCHAR(7) NOT NULL,
        correo_electronico VARCHAR(50) NOT NULL,syste
        tipo_inscripcion ENUM("ASISTENTE","CONFERENCISTA", "TALLERISTA", "OTRO") NOT NULL,
        CONSTRAINT PK_codigo_inscipcion PRIMARY KEY (codigo_inscripcion),
        CONSTRAINT FK_codigo_evento FOREIGN KEY (codigo_evento) REFERENCES evento (codigo_evento),
        CONSTRAINT FK_correo_electronico FOREIGN KEY (correo_electronico) REFERENCES participante
(correo_electronico)
);
```

```
CREATE TABLE pago(
        codigo_pago INT NOT NULL AUTO_INCREMENT,
        codigo_evento VARCHAR(7) NOT NULL,
        correo_electronico VARCHAR(50) NOT NULL,
        metodo_pago ENUM("EFECTIVO", "TRANSFERENCIA", "TARJETA") NOT NULL,
        monto DOUBLE NOT NULL,
        CONSTRAINT PK_codigo_pago PRIMARY KEY(codigo_pago),
        CONSTRAINT FK_codigo_evento2 FOREIGN KEY (codigo_evento) REFERENCES evento(codigo_evento),
        CONSTRAINT FK_correo_electronico FOREIGN KEY (correo_electronico) REFERENCES
participante(correo_electronico)
);

CREATE TABLE actividad(
        codigo_actividad VARCHAR(7) NOT NULL,
        codigo_evento VARCHAR(7) NOT NULL,
        correo_electronico VARCHAR(50) NOT NULL,
        tipo_actividad ENUM("CHARLA", "TALLER", "DEBATE", "OTRO") NOT NULL,
        titulo_actividad VARCHAR(200) NOT NULL,
        hora_inicio TIME NOT NULL,
        hora_fin TIME NOT NULL,
        cupo_maximo INT NOT NULL,
        CONSTRAINT PK_codigo_actividad PRIMARY KEY (codigo_actividad),
        CONSTRAINT FK_codigo_evento3 FOREIGN KEY (codigo_evento) REFERENCES evento (codigo_evento),
        CONSTRAINT FK_correo_electronico3 FOREIGN KEY (correo_electronico) REFERENCES participante
(correo_electronico)
);

CREATE TABLE asistencia(
        codigo_actividad VARCHAR(7) NOT NULL,
        correo_electronico VARCHAR(50) NOT NULL,
        CONSTRAINT FK_codigo_actividad4 FOREIGN KEY (codigo_actividad) REFERENCES actividad (codigo_actividad),
        CONSTRAINT FK_correo_electronico4 FOREIGN KEY (correo_electronico) REFERENCES participante
(correo_electronico) );
```

```
CREATE TABLE certificado(

        codigo_evento VARCHAR(7) NOT NULL,

        correo_electronico VARCHAR(50) NOT NULL,

        CONSTRAINT FK_codigo_evento5 FOREIGN KEY (codigo_evento) REFERENCES evento (codigo_evento),

        CONSTRAINT FK_correo_electronico5 FOREIGN KEY (correo_electronico) REFERENCES participante
(correo_electronico)

);
```

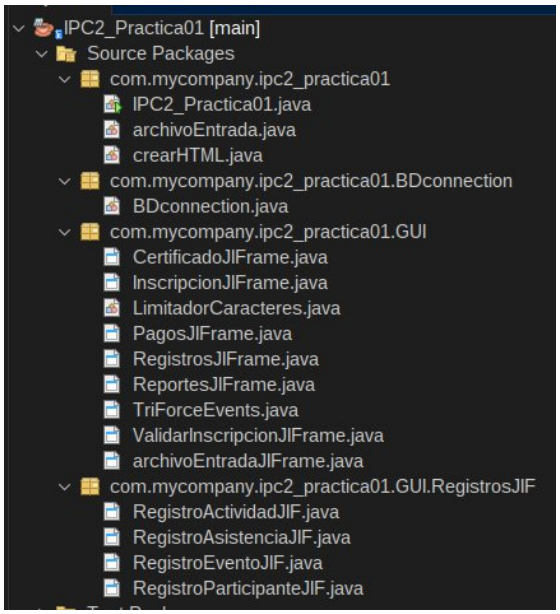
Descripción general del sistema:

El sistema de administración de eventos del Reino de Hyrule es una aplicación de escritorio desarrollada en Java, utilizando Swing para la construcción de la interfaz gráfica y MySQL como gestor de base de datos para el almacenamiento seguro y persistente de la información. Su diseño permite gestionar de forma integral los procesos relacionados con la organización de eventos, desde el registro de eventos y participantes, hasta la asignación de actividades, procesamiento de pagos y validación de inscripciones.

La aplicación ofrece al usuario la posibilidad de cargar archivos de entrada con instrucciones estructuradas, así como realizar las mismas operaciones a través de formularios amigables dentro de una interfaz basada en JDesktopPane, que facilita la navegación entre múltiples ventanas internas.

Entre sus principales funcionalidades se incluyen controles de cupo en actividades, verificación de pagos, prevención de inscripciones duplicadas y emisión automática de certificados para los participantes que cumplan los requisitos. Además, el sistema genera reportes en formato HTML, los cuales se guardan en la carpeta de salida definida por el usuario, permitiendo una consulta rápida y organizada de la información.

La forma en la cual esta estructurado el proyecto es la siguiente:



A continuación, se detallaran las principales clases que forman parte del sistema, sus roles, y como se interconectan para cumplir con los objetivos del software:

PAQUETE PRINCIPAL

1. IPC2_Practica01.java: Clase principal donde se encuentra el main y donde se instancia el JFrame principal de la aplicación:

```java
public class IPC2_Practica01 {

    public static void main(String[] args) {
        System.out.println("Practica 1 IPC2");
        TriForceEvents gui = new TriForceEvents();
        gui.setVisible(true);
    }
}
```

2. Archivo Entrada.java: Clase en la cual se leerá un archivo de texto, cuenta con métodos para interpretar las instrucciones del archivo y ejecutar y/o ignorarlas.

2.1 leerArchivo(File archivo):

```java
/**
 * METODO QUE LEE EL ARCHIVO DE TEXTO CON LAS INSTRUCCIONES PARA HACER LOS INSERTS
 * Y CADA LINEA LAS COLOCA EN UN ARRAYLIST PARA PODER ANALIZARLOS DE MEJOR MANERA
 * @param archivo
 */
public void leerArchivo(File archivo){
    ArrayList<String> lineas = new ArrayList<>();
    try (BufferedReader entrada = new BufferedReader(new FileReader(archivo))){
        String linea;
        while((linea = entrada.readLine()) != null){
            lineas.add(linea);
        }
    } catch (IOException e) {
        System.out.println("Error al leer el archivo");
    }

    interprete(lineas);
}
```

2.2 interprete(ArrayList<String> contenido):

```java
/**
 * METODO QUE LEE LINEA POR LINEA PARA SABER QUE TIPO DE INSTRUCCION HAY QUE REALIZAR
 * ASI MISMO SEPARA EL TIPO DE INSTRUCCION DE LOS PARAMETROS DE LA INSTRUCCION
 * @param contenido
 */
private void interprete(ArrayList<String> contenido){
    String instruccion;
    String parametros;
    for(String linea: contenido){

        if(linea == null || linea.trim().isEmpty()){
            continue;
        }

        int inicio = linea.indexOf("(");
        int fin = linea.indexOf(")");


        if (inicio != -1 && fin != -1 && fin > inicio) {
            instruccion = linea.substring(0, inicio).trim();

            parametros = linea.substring(inicio + 1, fin).trim();

            tipoInsert(instruccion, parametros);

        } else {
            System.out.println("No se puede hacer esta instruccion");
        }
    }
}
```

2.3 tipoInsert(String tipo, String parametros):

```java
/**
 * RECONOCE QUE TIPO DE INSERT HAY QUE HACER PARA SUBIR LA INFORMACION A LA BASE DE DATOS
 * Y MANDA A LLAMAR EL METODO QUE SE ENCARGA DE HACER EL INSERT DEPENDIENDO DE LA INSTRUCCION
 * @param tipo
 * @param parametros
 */
private void tipoInsert(String tipo, String parametros){
    switch(tipo){
        case "REGISTRO_EVENTO":
            insertEvento(separarParametros(parametros));
            break;
        case "REGISTRO_PARTICIPANTE":
            insertParticipante(separarParametros(parametros));
            break;
        case "INSCRIPCION":
            insertInscripcion(separarParametros(parametros));
            break;
        case "PAGO":
            insertPago(separarParametros(parametros));
            break;
        case "VALIDAR_INSCRIPCION":
            insertValidarInscrip(separarParametros(parametros));
            break;
        case "REGISTRO_ACTIVIDAD":
            insertActividad(separarParametros(parametros));
            break;
        case "ASISTENCIA":
            insertAsistencia(separarParametros(parametros));
            break;
        case "CERTIFICADO":
            insertCertificado(separarParametros(parametros));
            break;
        default:
            break;
    }
}
```

2.4 separarParametros(String parametros):

```java
/**
 * METODO QUE SEPARA LOS PARAMETROS PARA PODER ANALIZARLOS UNO POR UNO
 * SI SE ENCUENTRA ALGUN PARAMETRO VACIO, HARA SE DEVUELVA UN VALOR NULL
 * @param parametros
 * @return parametrosList
 */
private String[] separarParametros(String parametros) {
    String[] parametrosList = parametros.split(",");

    for (int i = 0; i < parametrosList.length; i++) {
        String parametro = parametrosList[i].trim();

        if (parametro.equals("\"\"")) {
            return null;
        } else if (parametro.startsWith("\"") && parametro.endsWith("\"")) {
            parametrosList[i] = parametro.substring(1, parametro.length() - 1);
            if (parametrosList[i].isEmpty()) {
                return null;
            }
        } else if (parametro.isEmpty()) {
            return null;
        }
    }

    return parametrosList;
}
```

2.5 Inserts: Luego de separar la instrucción de los parámetros y realizar algunas validaciones, se mandara a llamar a los métodos para poder hacer los INSERTs correspondientes estos métodos están en la clase BDconnection

Insert para evento:

```java
/**
 * METODO QUE SE ENCARGA DE HACER EL INSERT PARA EL EVENTO
 * SI LA LISTA DE PARAMETROS ES NULL, ESTO QUIERE DECIR QUE
 * LOS PARAMETROS ESTAN INCOMPLETOS ENTONCES SE OMITARA
 * @param parametros
 */
private void insertEvento(String[] parametros){
    if(parametros == null){
        System.out.println("No tiene todos los parametros");
        return;
    }

    String codEvento = parametros[0];
    String fecha = parametros[1];
    String tipoEvento = parametros[2];
    String tituloEvento = parametros[3];
    String ubicacion = parametros[4];
    int cupoMax = Integer.parseInt(parametros[5]);
    double costo = Double.parseDouble(parametros[6]);



    con.registrarEvento(codEvento, fecha, tipoEvento, tituloEvento, ubicacion, cupoMax, costo);
}
```

3. crearHTML: Clase en la cual se crearan los respectivos reportes y certificados en formato HTML gracias a un BufferedWriter

```java
/**
 * METODO QUE HACE EL CERTIFICADO DEL PARTICIPANTE EN FORMATO HTML
 * @param ruta
 * @param nombreDelArchivo
 * @param nombre
 * @param nombreEvento
 */
public void emitirCertificado(String ruta, String nombreDelArchivo, String nombre, String nombreEvento) {
    File archivo = new File(ruta, nombreDelArchivo);

    try (BufferedWriter salida = new BufferedWriter(new FileWriter(archivo))) {
        salida.write("<html>\n");
        salida.write("<head>\n");
        salida.write("<title>CERTIFICADO</title>\n");
        salida.write("</head>\n");
        salida.write("<body>\n");

        salida.write("<h1 style='text-align:center; font-size:64px;'>TriForce Events</h1>\n");
        salida.write("<h1 style='text-align:center; font-size:48px;'>CERTIFICADO</h1>\n");
        salida.write("<p style='text-align:center; font-size:24px;'>Se certifica que el Sr:</p>\n");
        salida.write("<h2 style='text-align:center; font-size:36px;'>" + nombre + "</h2>\n");
        salida.write("<p style='text-align:center; font-size:24px;'>Ha completado satisfactoriamente la actividad/evento:</p>\n");
        salida.write("<h3 style='text-align:center; font-size:36px;'>" + nombreEvento + "</h3>\n");

        salida.write("<p style='text-align:center;'>_____</p>");
        salida.write("<p style='text-align:center; font-size:36px;'>Firma</p>");

        salida.write("</body>\n");
        salida.write("</html>\n");
    } catch (IOException e) {
        System.out.println("HA OCURRIDO UN ERROR AL CREAR CERTIFICADO");
    }
}
```

PAQUETE BDconnection

1. BDconnection: Clase principal del sistema, donde se realizan los INSERTs y las validaciones para hacer los mismos, para poder ingresar los datos a la base de datos

1.1 connect() : Método en el cual se conecta la aplicación a la base de datos

```
private static final String IP = "localhost";
private static final int PUERTO = 3306;
private static final String SCHEMA = "TriForceEvents";
private static final String USER_NAME = "adminDBA";
private static final String PASSWORD = "admin@123";

private static final String URL = "jdbc:mysql://" + IP + ":" + PUERTO + "/" + SCHEMA;

private Connection connection;

private crearHTML html = new crearHTML();
private String ruta = "/home/jgarcia07/NetBeansProjects/IPC2_Practica01/Reportes";

/**
 * METODO PARA CONECTARSE A LA BASE DE DATOS
 */
public void connect() {
    try {
        connection = DriverManager.getConnection(URL, USER_NAME, PASSWORD);
        System.out.println("CONEXION EXITOSA A LA BASE DE DATOS");
    } catch (Exception e) {
        System.out.println("ERROR AL CONECTARSE");
        e.printStackTrace();
    }
}
```

1.2 buscarCodigo( ):

```
/**
 * FUNCION QUE BUSCARA UN CODIGO EN UNA TABLA DE LA BASE DE DATOS PARA VERIFICAR SI
 * EXISTE, YA SEA PARA VALIDAR QUE SE PUEDA HACER EL INSERT O BIEN QUE EL INSERT NO
 * SEA REPETIDO
 * @param codigo
 * @param tabla
 * @param tipoCodigo
 * @return
 */
private boolean buscarCodigo(String codigo, String tabla, String tipoCodigo) {
    boolean existeElCodigo = false;
    String sqlCodigo = "SELECT COUNT(*) FROM " + tabla + " WHERE " + tipoCodigo + " = ?";
    try (PreparedStatement psCodigo = connection.prepareStatement(sqlCodigo)) {
        psCodigo.setString(1, codigo);
        ResultSet rs = psCodigo.executeQuery();
        rs.next();
        if (rs.getInt(1) > 0) {
            existeElCodigo = true;
        }
    } catch (SQLException e) {
        JOptionPane.showMessageDialog(null, "Ha ocurrido un error inesperado");
    }

    return existeElCodigo;
}
```

1.3 buscarEmail( ):

```java
/**
 * FUNCION  QUE BUSCARA UN EMAIL EN UNA TABLA DE LA BASE DE DATOS PARA VERIFICAR SI
 * EXISTE, YA SEA PARA VALIDAR QUE SE PUEDA HACER EL INSERT O BIEN QUE EL INSERT NO
 * SEA REPETIDO
 * @param email
 * @param tabla
 * @return
 */
private boolean buscarEmail(String email, String tabla){
    boolean existeElEmail = false;
    String sqlEmail = "SELECT COUNT(*) FROM "+ tabla + " WHERE correo_electronico = ?";
    try (PreparedStatement psEmail = connection.prepareStatement(sqlEmail)){
        psEmail.setString(1,email);
        ResultSet rs = psEmail.executeQuery();
        rs.next();
        if(rs.getInt(1) > 0){

            existeElEmail = true;
        }
    } catch (SQLException e) {
        JOptionPane.showMessageDialog(null, "Ha ocurrido un error inseperado");
    }
    return existeElEmail;
}
```

1.4 validarCorreoYcodEvento ( ):

```java
/**
 * FUNCION QUE BUSCARA UN CODIGO Y UN EMAIL EN UNA TABLA DE LA BASE DE DATOS PARA VERIFICAR SI
 * EXISTE, YA SEA PARA VALIDAR QUE SE PUEDA HACER EL INSERT O BIEN QUE EL INSERT NO
 * SEA REPETIDO
 * @param email
 * @param codEvento
 * @param tabla
 * @return
 */
private boolean validarCorreoYcodEvento(String email, String codEvento, String tabla){
    boolean existeRegistro = false;
    String sqlValidacion = "SELECT codigo_evento, correo_electronico FROM "+tabla+" WHERE codigo_evento = ? AND correo_electronico = ?";
    try (PreparedStatement psValidar = connection.prepareStatement(sqlValidacion)){
        psValidar.setString(1, codEvento);
        psValidar.setString(2, email);
        ResultSet rs = psValidar.executeQuery();
        if(rs.next()){
            existeRegistro = true;
        }
    } catch (SQLException e) {
        System.out.println("Ha ocurrido un error inseperado");
        e.printStackTrace();
    }

    return existeRegistro;
}
```

Una vez tenemos las validaciones ya podemos ver los Inserts:

```java
/**
 * METODO QUE REALIZA EL INSERT PARA EL EVENTO, CON SUS RESPECTIVAS VALIDACIONES
 * PARA EVITAR EXCEPCIONES Y SE UTILIZA EL PreparedStatement PARA EVITAR EL SQL
 * INYECTION
 * @param codEvento
 * @param fecha
 * @param tipoEvento
 * @param tituloEvento
 * @param ubicacion
 * @param cupoMax
 * @param costo
 */
public int registrarEvento(String codEvento, String fecha, String tipoEvento, String tituloEvento, String ubicacion, int cupoMax, double costo) {
    //HAY QUE VERIFICAR SI EL CODIGO DE EVENTO YA EXISTE
    if (buscarCodigo(codEvento, "evento", "codigo_evento") == true) {
        return 1;
    }

    String sql = "INSERT INTO evento (codigo_evento, fecha_evento, tipo_evento, titulo_evento, ubicacion, cupo_maximo, costo_inscripcionn) VALUES (?, ?, ?, ?, ?, ?, ?)";
    try (PreparedStatement ps = connection.prepareStatement(sql)) {
        //PASAR DE D/M/A -> A/M/D
        SimpleDateFormat date = new SimpleDateFormat("dd/MM/yyyy");
        java.util.Date dateUtil = date.parse(fecha);
        java.sql.Date dateSQL = new java.sql.Date(dateUtil.getTime());

        ps.setString(1, codEvento);
        ps.setDate(2, dateSQL);
        ps.setString(3, tipoEvento);
        ps.setString(4, tituloEvento);
        ps.setString(5, ubicacion);
        ps.setInt(6, cupoMax);
        ps.setDouble(7, costo);

        int rowsAffected = ps.executeUpdate();
        return mensajeQuery(rowsAffected);
    } catch (SQLException e) {
        JOptionPane.showMessageDialog(null, "Ha ocurrido un error inseperado");
        e.printStackTrace();
        return -1;
    } catch (ParseException ex) {
        JOptionPane.showMessageDialog(null, "Formato invalido de hora");
        ex.printStackTrace();
        return -1;
    }
}
```

```java
/**
 * METODO QUE REALIZA EL INSERT PARA EL PARTICIPANTE, CON SUS RESPECTIVAS VALIDACIONES
 * PARA EVITAR EXCEPCIONES Y SE UTILIZA EL PreparedStatement PARA EVITAR EL SQL
 * INYECTION
 * @param nombre
 * @param tipoParticipante
 * @param institucion
 * @param email
 */
public int registrarParticipante(String nombre, String tipoParticipante, String institucion, String email) {
    //HAY QUE VALIDAR SI YA EXISTE EL USUARIO
    if(buscarEmail(email, "participante") == true){
        return 1;
    }

    String sql = "INSERT INTO participante (nombre_participante, rol_participante, correo_electronico, institucion_procedencia) VALUES (?, ?, ?, ?)";
    try (PreparedStatement ps = connection.prepareStatement(sql)) {
        ps.setString(1, nombre);
        ps.setString(2, tipoParticipante);
        ps.setString(3, email);
        ps.setString(4, institucion);

        int rowsAffected = ps.executeUpdate();
        return mensajeQuery(rowsAffected);
    } catch (SQLException e) {
        JOptionPane.showMessageDialog(null,"Ha ocurrido un error inseperado");
        e.printStackTrace();
        return -1;
    }
}
```

```java
/**
 * METODO QUE REALIZA EL INSERT PARA LA INSCRIPCION, CON SUS RESPECTIVAS VALIDACIONES
 * PARA EVITAR EXCEPCIONES Y SE UTILIZA EL PreparedStatement PARA EVITAR EL SQL
 * INYECTION
 * @param email
 * @param codEvento
 * @param tipoInscripcion
 */
public int inscripcion(String email, String codEvento, String tipoInscripcion) {

    if(buscarCodigo(codEvento, "evento", "codigo_evento") == false){
        return 1;
    } else if(buscarEmail(email, "participante") == false){
        return 2;
    } else if(validarCorreoYcodEvento(email, codEvento, "inscripcion") == true){
        return 3;
    }

    String sql = "INSERT INTO inscripcion (codigo_evento, correo_electronico, tipo_inscripcion) VALUES (?, ?, ?)";
    try (PreparedStatement ps = connection.prepareStatement(sql)) {
        ps.setString(1, codEvento);
        ps.setString(2, email);
        ps.setString(3, tipoInscripcion);

        int rowsAffected = ps.executeUpdate();
        return mensajeQuery(rowsAffected);
    } catch (SQLException e) {
        System.out.println("Ha ocurrido un error inseperado");
        e.printStackTrace();
        return -1;
    }
}
```

```java
/**
 * METODO QUE REALIZA EL INSERT PARA EL PAGO, CON SUS RESPECTIVAS VALIDACIONES
 * PARA EVITAR EXCEPCIONES Y SE UTILIZA EL PreparedStatement PARA EVITAR EL SQL
 * INYECTION
 * @param email
 * @param codEvento
 * @param metodoPago
 * @param monto
 */
public int pago(String email, String codEvento, String metodoPago, double monto) {
    double montoBD = 0;

    if(buscarEmail(email , "participante") == false){
        return 1;
    } else if(buscarCodigo(codEvento, "evento", "codigo_evento") == false){
        return 2;
    } else if(validarCorreoYcodEvento(email, codEvento, "pago") == true){
        return 3;
    }

    String sqlValidacionMonto = "SELECT costo_inscripcionn FROM evento WHERE codigo_evento = ?";
    try (PreparedStatement psMonto = connection.prepareStatement(sqlValidacionMonto)){
        psMonto.setString(1, codEvento);
        ResultSet rs = psMonto.executeQuery();
        if(rs.next()){
            montoBD = rs.getDouble("costo_inscripcionn");
        }
    } catch (SQLException e) {
        JOptionPane.showMessageDialog(null, "Ha ocurrido un error inesperado al buscar el codigo de evento");
        return -1;
    }

    if(montoBD == monto){
        String sql = "INSERT INTO pago (codigo_evento, correo_electronico, metodo_pago, monto) VALUES (?, ?, ?, ?)";
        try (PreparedStatement ps = connection.prepareStatement(sql)) {
            ps.setString(1, codEvento);
            ps.setString(2, email);
            ps.setString(3, metodoPago);
            ps.setDouble(4, monto);

            int rowsAffected = ps.executeUpdate();
            return mensajeQuery(rowsAffected);
        } catch (SQLException e) {
            System.out.println("Ha ocurrido un error inseperado");
            e.printStackTrace();
            return -1;
        }
    } else {
        JOptionPane.showMessageDialog(null, "El monto a pagar no es lo que equivale el costo del evento");
        return -1;
    }
}
```

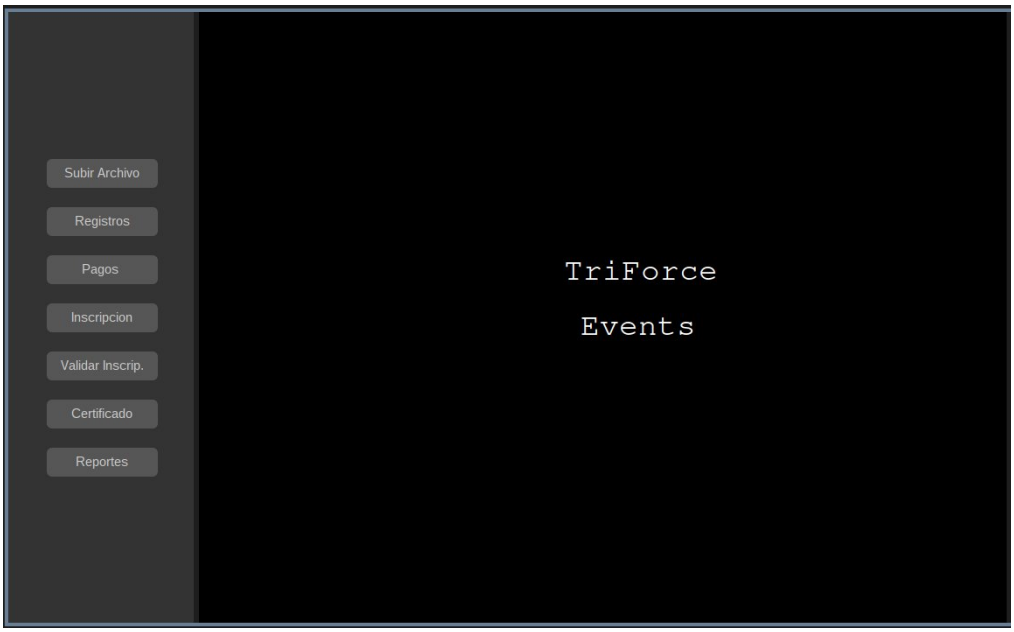
Reportes:

```java
public int reporteParticipantes(){
    String sql = "SELECT * FROM participante";
    try (PreparedStatement ps = connection.prepareStatement(sql)){
        ResultSet rs = ps.executeQuery();

        String nombreArchivo =  "Reporte De Participantes.html";

        html.reporte(ruta, nombreArchivo, "Reporte de Participante",rs);
        return 0;
    } catch (SQLException e) {
        e.printStackTrace();
        return -1;
    }
}

public int reporteActividades(){
    String sql = "SELECT * FROM actividad";
    try (PreparedStatement ps = connection.prepareStatement(sql)){
        ResultSet rs = ps.executeQuery();

        String nombreArchivo =  "Reporte De Actividades.html";

        html.reporte(ruta, nombreArchivo, "Reporte de Actividades",rs);
        return 0;
    } catch (SQLException e) {
        e.printStackTrace();
        return -1;
    }
}

public int reporteEventos(){
    String sql = "SELECT * FROM evento";
    try (PreparedStatement ps = connection.prepareStatement(sql)){
        ResultSet rs = ps.executeQuery();

        String nombreArchivo =  "Reporte De Eventos.html";

        html.reporte(ruta, nombreArchivo, "Reporte de Eventos",rs);
        return 0;
    } catch (SQLException e) {
        e.printStackTrace();
        return -1;
    }
}
```

PAQUETE GUI

Paquete donde como su nombre lo menciona se encuentra la interfaz de usuario, para este parte se utilizo un JFrame principal llamado TriForceEvents, a este se le coloco un JDesktop Pane, las demás funciones se hicieron en JInterFrames donde se movían gracias al JDesktop Pane.

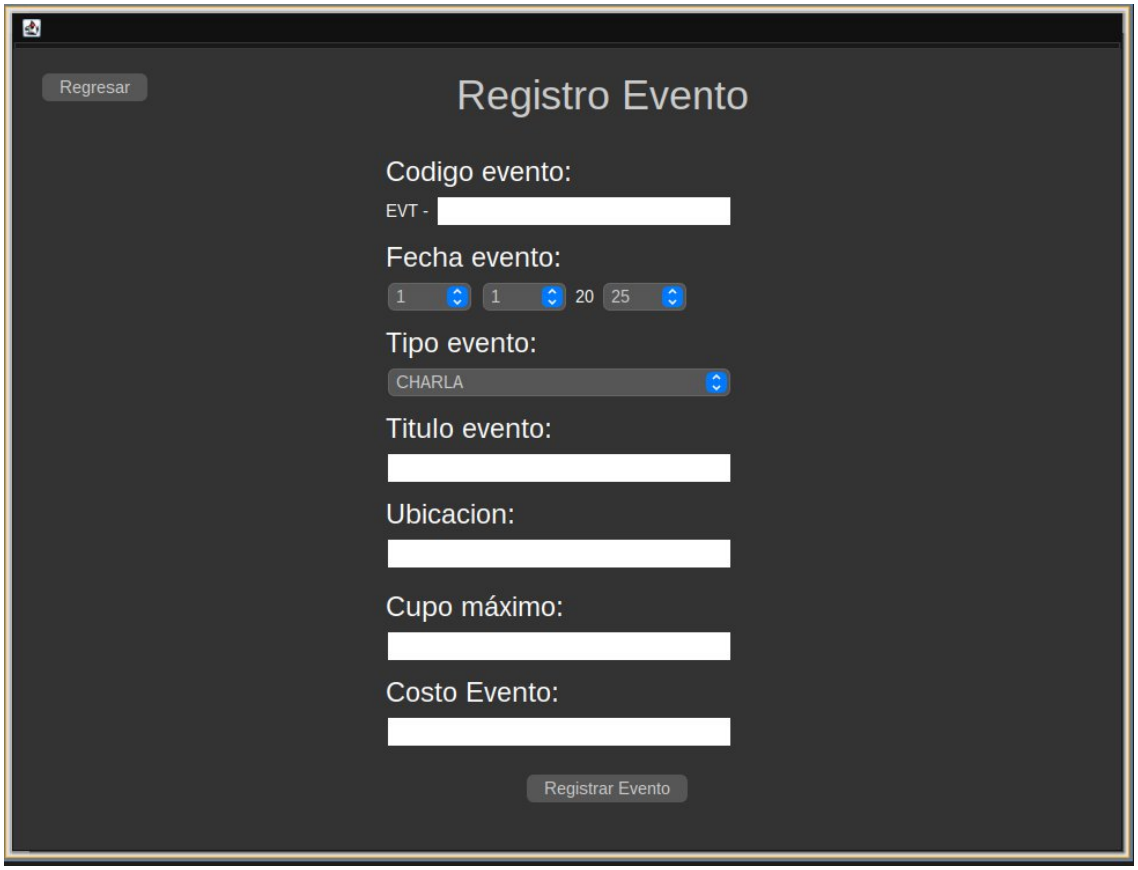
- TriForceEvents:



En los botones se mandan a llamar a los JInternalFrame de la siguiente manera:

```
private void btn_RegistrosActionPerformed(java.awt.event.ActionEvent evt) {
    RegistrosJIFrame registros = new RegistrosJIFrame();
    MenuPrincipal.add(registros);
    registros.setVisible(true);
}

private void btn_InscripcionActionPerformed(java.awt.event.ActionEvent evt) {
    InscripcionJIFrame inscripcion = new InscripcionJIFrame();
    MenuPrincipal.add(inscripcion);
    inscripcion.setVisible(true);
}

private void btn_ReportesActionPerformed(java.awt.event.ActionEvent evt) {
    ReportesJIFrame reporte = new ReportesJIFrame();
    MenuPrincipal.add(reporte);
    reporte.setVisible(true);
}

private void btn_PagosActionPerformed(java.awt.event.ActionEvent evt) {
    PagosJIFrame pagos = new PagosJIFrame();
    MenuPrincipal.add(pagos);
    pagos.setVisible(true);
}

private void btn_CertificadoActionPerformed(java.awt.event.ActionEvent evt) {
    CertificadoJIFrame certificado = new CertificadoJIFrame();
    MenuPrincipal.add(certificado);
    certificado.setVisible(true);
}

private void btn_validarInscripcionActionPerformed(java.awt.event.ActionEvent evt) {
    ValidarInscripcionJIFrame validarInscrip = new ValidarInscripcionJIFrame();
    MenuPrincipal.add(validarInscrip);
    validarInscrip.setVisible(true);
}

private void btn_subirArchivoActionPerformed(java.awt.event.ActionEvent evt) {
    archivoEntradaJIFrame archivo = new archivoEntradaJIFrame();
    MenuPrincipal.add(archivo);
    archivo.setVisible(true);
}
```

Dependiendo de casa registro tendrá diferentes valores pero la mayoría de los JInternalFrame tienen la siguiente estructura:



Todos los TextFields tienen sus respectivos limitadores de caracteres para evitar problemas con la base de datos, así mismo se valida que llene todos los campos. Y la manera de enviar los datos a la base de datos es la siguiente:

```
private void btn_registrarEventoActionPerformed(java.awt.event.ActionEvent evt) {
    final String preFijoEvt = "EVT-";
    if (TF_codEvento.getText().trim().isEmpty() || TF_titulo.getText().trim().isEmpty() || TF_ubicacion.getText().trim().isEmpty()
            || TF_cupoMax.getText().trim().isEmpty() || TF_costo.getText().trim().isEmpty()) {
        JOptionPane.showMessageDialog(null, "Debe de llenar todos los campos");
    } else {
        try {
            String codEvento = preFijoEvt + TF_codEvento.getText();
            String fecha = CB_dia.getSelectedItem().toString() + "/" + CB_mes.getSelectedItem().toString() + "/20" + CB_año.getSelectedItem().toString();
            String tipoEvento = CB_tipoEvento.getSelectedItem().toString();
            String titulo = TF_titulo.getText();
            String ubicacion = TF_ubicacion.getText();
            int cupoMax = parseInt(TF_cupoMax.getText());
            double costo = parseDouble(TF_costo.getText());

            BDconnection con = new BDconnection();
            con.connect();

            int respuesta = con.registrarEvento(codEvento, fecha, tipoEvento, titulo, ubicacion, cupoMax, costo);

            if(respuesta == 0){
                JOptionPane.showMessageDialog(null,"Datos ingresador correctamente");
            } else if (respuesta  == 1) {
                JOptionPane.showMessageDialog(null, "El codigo el evento ya existe en el registro");
            }

            TF_codEvento.setText("");
            TF_costo.setText("");
            TF_cupoMax.setText("");
            TF_titulo.setText("");
            TF_ubicacion.setText("");

        } catch (NumberFormatException e) {
            JOptionPane.showMessageDialog(null, "Deben de ser valores numericos el Cupo Máximo y el Costo");
        }
    }
}
```

Este proyecto, desarrollado con MySQL como motor de base de datos, Java como lenguaje de programación y Swing para la interfaz gráfica, ha sido diseñado para gestionar de manera eficiente la organización de eventos. La estructura de la base de datos, detallada en el diagrama ER, es fundamental para la correcta funcionalidad del sistema.

Las tablas principales, como Participante, Evento y Actividad, están lógicamente conectadas para manejar las inscripciones, pagos, y la emisión de certificados.

El uso de MySQL asegura la persistencia de los datos, permitiendo un almacenamiento seguro y estructurado de la información. La integración con Java/Swing facilita la interacción del usuario con la base de datos a través de una interfaz gráfica intuitiva y amigable.

La robustez del modelo de datos garantiza que el sistema pueda escalar, registrando de manera precisa la participación en actividades, los pagos asociados y la validación de la asistencia. En resumen, la combinación de estas tecnologías permite una solución integral y fiable para la gestión de eventos, desde la inscripción inicial hasta la emisión de certificados.