

# **MANUAL TÉCNICO**

Liminalis: Juego de Trivia

**Lenguaje:** Java

**Base de Datos:** SQLite

## **Autores:**

Heidy Cruz

Ángela Cevallos

Ismael Heredia

Andrian Muñoz

Mathias Cruz

## **Institución:**

Escuela Politécnica Nacional

28 de enero de 2026

# Índice general

<b>1. ¿Qué es Liminalis?</b>	<b>3</b>
<b>2. Alcance del Sistema</b>	<b>4</b>
2.1. Usuario de tipo jugador . . . . .	4
2.2. Usuario de tipo administrador . . . . .	7
<b>3. Tecnologías Utilizadas</b>	<b>12</b>
<b>4. Arquitectura del Sistema</b>	<b>14</b>
4.1. Tipo de Arquitectura . . . . .	14
4.2. Diagrama de Arquitectura . . . . .	14
<b>5. Estructura del Proyecto</b>	<b>15</b>
<b>6. Base de Datos</b>	<b>19</b>
6.1. Sistema Gestor de Base de Datos (SGDB) . . . . .	19
6.2. Modelo Entidad Relación (MER) . . . . .	20
<b>7. Lógica del Sistema</b>	<b>21</b>
7.1. DataHelperSQLite . . . . .	21
7.2. Interfaz del Videojuego . . . . .	22
7.3. Métodos de interfaz . . . . .	22
7.4. Métodos de interfaz que regresan los páneles . . . . .	22
7.5. Uso de layouts . . . . .	23
7.6. ScreenKeyboard/Teclado en pantalla . . . . .	23
7.7. ScreenGame/Pantalla de Juego . . . . .	24
7.8. Relación de tablas . . . . .	24
<b>8. Descripción de Clases</b>	<b>25</b>

<b>9. Manejo de Errores</b>	<b>28</b>
9.1. Arquitectura del Manejo de Errores . . . . .	28
9.2. Clase Base de Excepciones . . . . .	29
9.2.1. Tipos de Excepciones . . . . .	29
9.3. Sistema de Registro de Errores . . . . .	29
9.3.1. Ubicación de los Archivos de Log . . . . .	29
9.3.2. Formato del Registro . . . . .	29
9.4. Propagación de Excepciones . . . . .	30
9.5. Validaciones . . . . .	30
9.6. Buenas Prácticas . . . . .	30
<b>10. Instalación y Configuración</b>	<b>31</b>

# Capítulo 1

## ¿Qué es Liminalis?

Liminalis, es un juego de trivia enfocado en desafiar a todos los jugadores con preguntas de diferentes categorías: planetas, cultura general, geografía, videojuegos, entre otras. El jugador puede registrarse con el alias que prefiera y empezar a jugar para competir en el marcador de los mayores puntajes. La tecnología empleada para este juego, es el lenguaje de programación orientado a objetos Java y el gestor de Base de Datos SQLite. El juego cuenta con interfaz gráfica impulsada por la librería o API propia del lenguaje: Java Swing.



Figura 1.1: Logo del juego

# Capítulo 2

## Alcance del Sistema

### 2.1. Usuario de tipo jugador

La pantalla de bienvenida permite a los usuarios seleccionar las opciones de jugar, ver marcador, salir o acceder a la autenticación de administrador, cada opción en las pantallas presentadas en el juego pueden ser seleccionadas mediante un joystick, a través de las acciones determinadas con atajos para la comunicación entre el mando y el teclado, por lo cual si el usuario requiere digitar alguna información lo puede hacer mediante un teclado propio del juego.



Figura 2.1: Pantalla de bienvenida

Los jugadores registrados son mostrados con el botón "Marcador" que permite mostrar el panel de jugadores registrados.

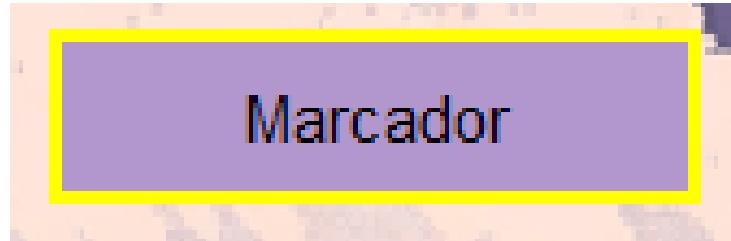


Figura 2.2: Botón de marcador de jugadores

El panel de jugadores registrados, muestra los campos del nombre del jugador, su puntuación y su última jugada.

The screenshot shows a modal window titled "Marcador De Jugadores". Inside the window, there is a table with the following data:

Id Jugador	Jugador	Puntuación	Última Jugada
2	GamerGirl	5000	2026-01-24 23:28...
7	Ism	5000	2026-01-24 23:44...
5	Brainiac	2500	
4	QuizMaster	2200	
3	Speedster	1800	
1	PlayerOne	1500	
9	gggg	30	2026-01-24 23:29...
11	Popeye	23	2026-01-25 01:49...
10	1hg	20	2026-01-24 23:41...
6	Bachily	10	2026-01-24 22:39...
12	FuncionaValidarEspacios	10	2026-01-25 01:47...
8	hmmm	0	2026-01-24 23:29...
13	JugadorNuevo	0	2026-01-25 02:59...

At the bottom right of the window, there is an "OK" button.

Figura 2.3: Marcador de jugadores

El juego habilita al usuario a crear jugadores con un nombre asignado por preferencia, empezando con un puntaje inicial el cual irá incrementando conforme el jugador acumule puntos en cada sesión de juego. El registro responde únicamente a un nombre, no es necesario ingresar contraseña o iniciar sesión para jugar; el nombre que se ingrese corresponde a un jugador nuevo o ya creado previamente, de esto modo, si el jugador ya existe podrá continuar acumulando puntaje en sus nuevas partidas.



Figura 2.4: Pantalla de registro de jugador

Si el jugador ingresado es nuevo, empezara con un puntaje referencial de 0, y si es uno que se ha registrado previamente, podrá continuar jugando y ganando puntos.

	idPlay...	idUser...	Name	Score	Status	CreationDate	ModificateD...				
	Filter...	Filter...	Filter...	Filter...	Filter...	Filter...	Filter...				
1	1	1	PlayerOne	1500	Activo	2026-01-24 22:...	NULL				
2	2	1	GamerGirl	5000	Activo	2026-01-24 22:...	2026-01-24 23:...				
3	3	1	Speedster	1800	Activo	2026-01-24 22:...	NULL				
4	4	1	QuizMaster	2200	Activo	2026-01-24 22:...	NULL				
5	5	1	Brainiac	2500	Activo	2026-01-24 22:...	NULL				
6	6	1	Bachily	10	Activo	2026-01-24 22:...	2026-01-24 22:...				
7	7	1	Ism	5000	Activo	2026-01-24 23:...	2026-01-24 23:...				
8	8	1	hmmm	0	Activo	2026-01-24 23:...	2026-01-24 23:...				
9	9	1	gggg	30	Activo	2026-01-24 23:...	2026-01-24 23:...				
10	10	1	1hg	20	Activo	2026-01-24 23:...	2026-01-24 23:...				
11	11	1	Popeye	23	Activo	2026-01-25 01:...	2026-01-25 01:...				
12	12	1	FuncionaValidarEspacios	10	Activo	2026-01-25 01:...	2026-01-25 01:...				
13	13	1	JugadorNuevo	0	Activo	2026-01-25 02:...	2026-01-25 02:...				

Figura 2.5: Registros de jugadores en la BD

## 2.2. Usuario de tipo administrador

Además de los jugadores, existe otro tipo de usuarios los cuales son los administradores, quienes pueden gestionar los registros y modificar los campos relacionados con el jugador registrado, esto mediante un panel de gestión al cual se podrá acceder mediante una autenticación con usuario y contraseña como se muestra en la Figura 2.6

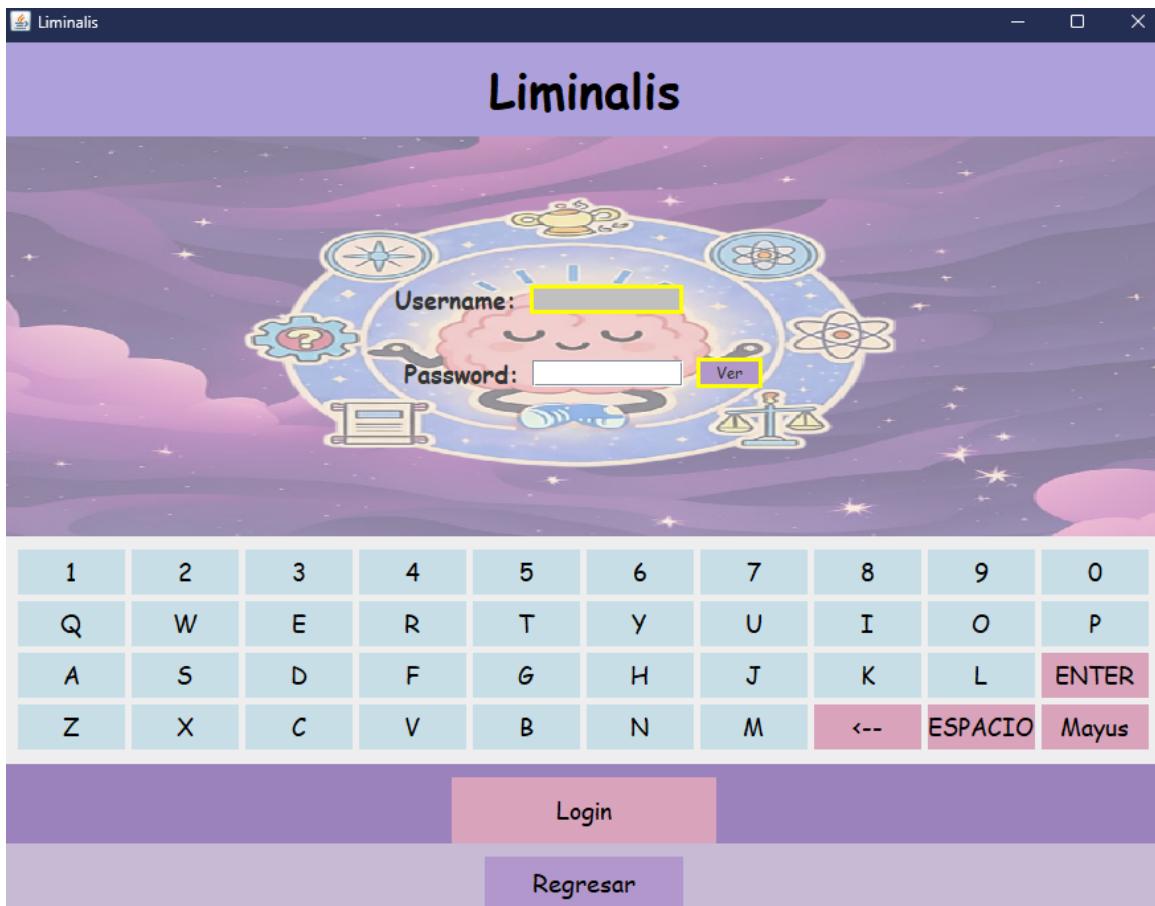


Figura 2.6: Pantalla de autenticación con usuario y contraseña

Una vez los administradores se autentiquen exitosamente, obtendrán acceso al panel de gestión que se observa en la Figura 2.7, donde el control de usuarios puede llevarse a cabo mediante la búsqueda, modificación y consulta de jugadores en la tabla, además de la posibilidad de regresar a la pantalla anterior de autenticación o salir directamente del juego.



Figura 2.7: Pantalla de gestión de usuarios

### ■ Tabla de jugadores

Permite observar los jugadores registrados en el sistema, mostrando su nombre de usuario con el que empezaron a jugar, su puntaje, su estado, fecha de creación y fecha de modificación en caso de que algún administrador haya modificado el registro.

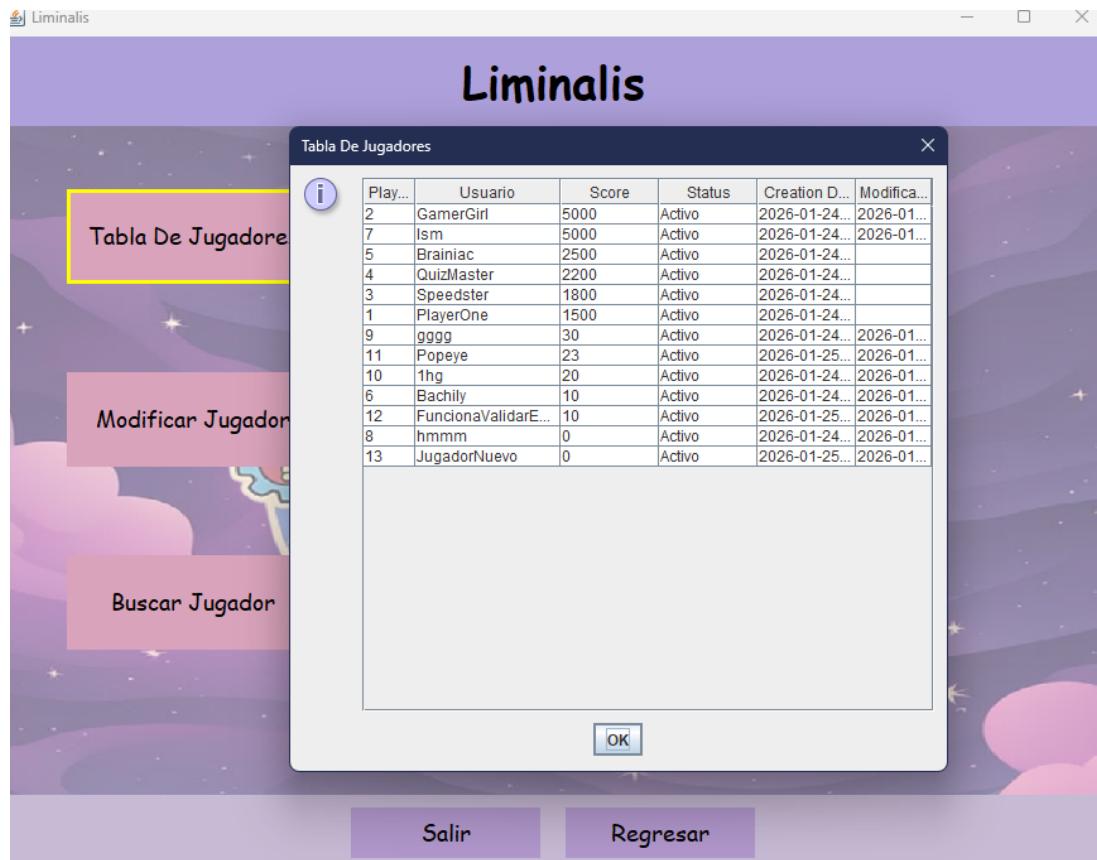


Figura 2.8: Tabla de usuarios registrados con sus datos correspondientes

### ■ Modificar Jugador

Permite modificar el nombre y puntaje de los jugadores registrados, además de mostrar toda la información del jugador para llevar un control mucho más exacto de los datos que requieren ser cambiados y el historial que ha tenido como jugador.

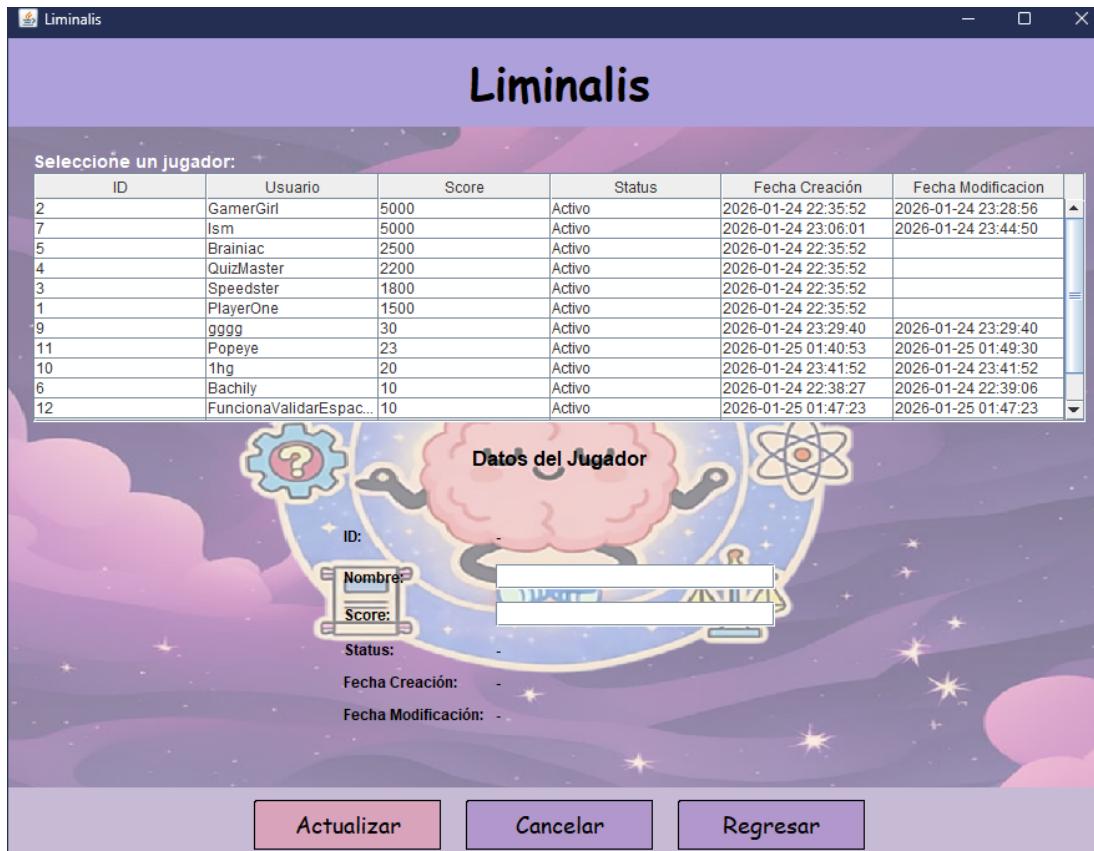


Figura 2.9: Pantalla de actualización de datos de jugadores

### ■ Buscar Jugador

Permite buscar un jugador por su ID en la base de datos o por el nombre único registrado, la navegación e ingreso de datos a buscar en los campos, se realiza mediante el joystick, con el cual se puede controlar las acciones a través de el teclado en pantalla.



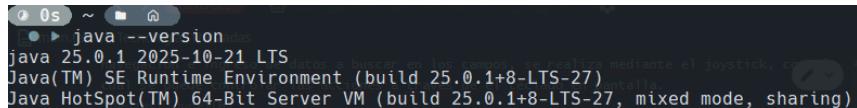
Figura 2.10: Pantalla para búsqueda de jugadores por id y nombre

# Capítulo 3

## Tecnologías Utilizadas

- **Lenguaje de programación**

Se utilizó el lenguaje de programación orientado a objetos Java con su versión 25.0.1, debido a lo estricto de la organización con la implementación de clases, su gestión de memoria, el empleo del paradigma POO y su extenso abanico de librerías propias que permiten acoplarse para las necesidades de cada proyecto.



```
OS ~ [ ] ⌂
  ➤ java --version
java 25.0.1 2025-10-21 LTS
Java(TM) SE Runtime Environment (build 25.0.1+8-LTS-27)
Java HotSpot(TM) 64-Bit Server VM (build 25.0.1+8-LTS-27, mixed mode, sharing)
```

Figura 3.1: Versión de Java

- **Interfaz Gráfica: Java Swing**

El empleo de la API/biblioteca Java Swing propia de Java, permitió el uso simple de pantallas gráficas que permitan interactuar al usuario final con las funcionalidades de la aplciación, gracias al manejo de clases dedicadas a los botones y estilos que tendra la aplicación en la presentación.

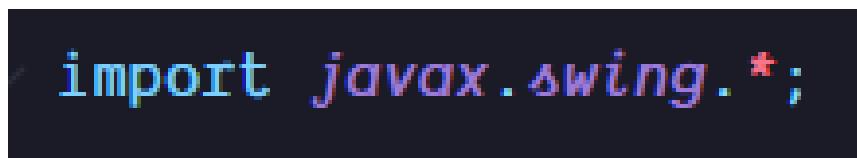


Figura 3.2: libreria JavaSwing

- **Base de Datos: SQLite**

Se utilizó el gestor de base de datos SQLite mediante el uso de extensiones propias del IDE VSCode, las cuales permiten la creación, inserción y manipulación de información tratada, además de posibilitar al programador a utilizar libremente las queries y visualizar el diseño de las tablas creadas en la base de datos.

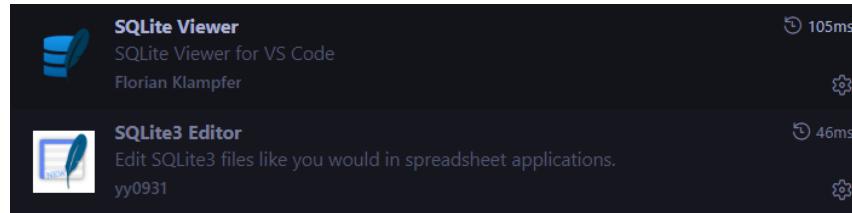


Figura 3.3: Extensiones SQLite

- **Conector: JDBC**

Se utilizó el conector JDBC versión 3.51.1.0, el cual permite la comunicación entre la base de datos SQL creada y la lógica del proyecto, considerando el uso de métodos, patrones de diseño, entre otros para la consulta de información.



Figura 3.4: Conector JDBC

- **Emulador DS4Windows**

Permite bindear las teclas del teclado con los botones, joystick y gatillos del mando, lo cual permitió la utilización del mando para jugar.



Figura 3.5: Aplicación para la emulación de mando

# Capítulo 4

## Arquitectura del Sistema

### 4.1. Tipo de Arquitectura

La Arquitectura empleada para el sistema del proyecto, fue la conocida Arquitectura N-Tier, basada en el manejo de capas para la realización de acciones relacionadas con patrones de diseño, conexión a las bases de datos, APIs, Frameworks, recursos o herramientas, librerías, entre otras formas de estructurar el proyecto de forma escalable.

### 4.2. Diagrama de Arquitectura

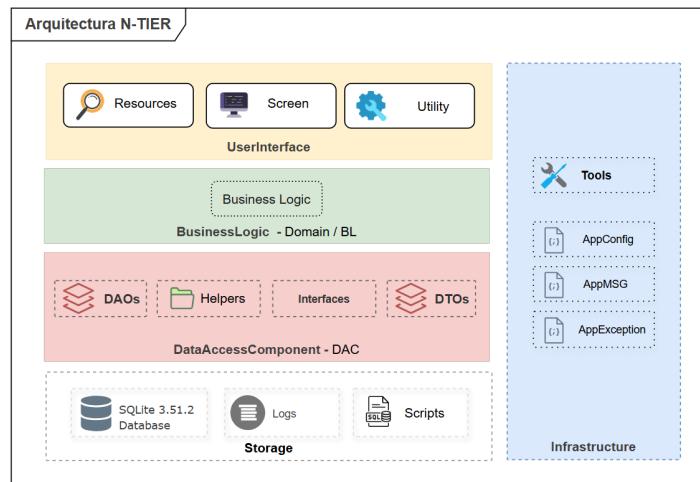


Figura 4.1: Arquitectura del sistema

# Capítulo 5

## Estructura del Proyecto

- **Directorio File**

Se guardan los requerimientos del sistema, casos de uso, diagrama de clase y modelos entidad relación.

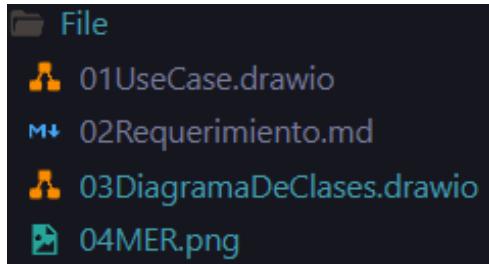


Figura 5.1: Directorio File

- **Directorio BussinesLogic**

Se encuentran las clases relacionadas con la lógica del negocio que se encuentran en el dominio de la aplicación, las cuales se encargarán de comunicar la capa de presentación o interfaz de usuario con los datos abstraídos en la capa DAC.

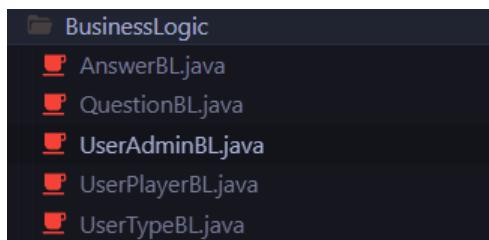


Figura 5.2: Directorio de BL

- **Directorio DataAccessComponent**

Almacena las clases encargadas de persistir la información abstraída de la base de datos, empleando las clases DAO y DTO, las cuales se encargan de realizar las consultas a la BD y tratar o guardar la información, respectivamente. Además, guarda las clases dedicadas a la conexión y comunicación con la base de datos utilizando el JDBC, y las interfaces que garanticen las acciones CRUD o métodos obligatorios de cada clase.

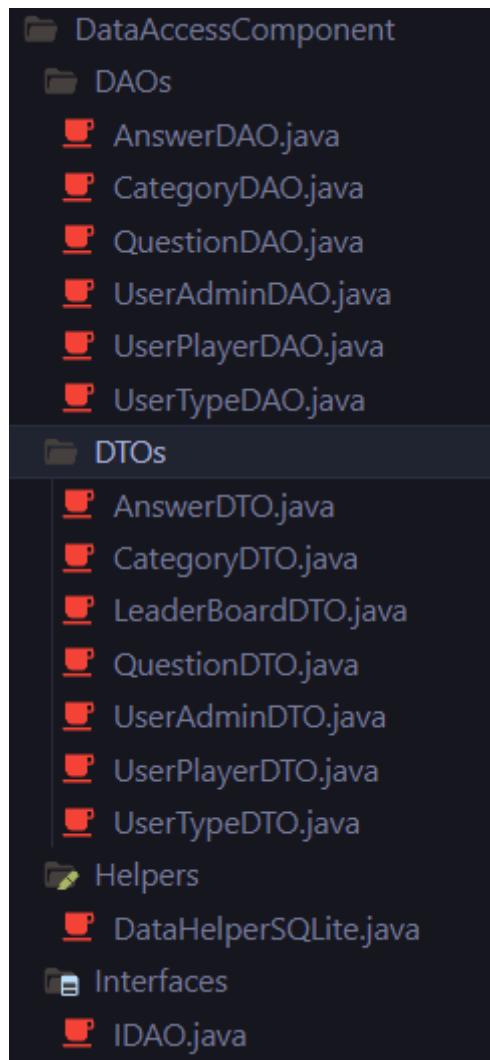


Figura 5.3: Directorio de DAC

- **Directorio Infrastructure**

Guarda las clases dedicadas a funcionar como herramientas para personalizar la terminal, gestionar el manejo de errores o funcionalidades adicionales que se decida darle al sistema a nivel de programación.

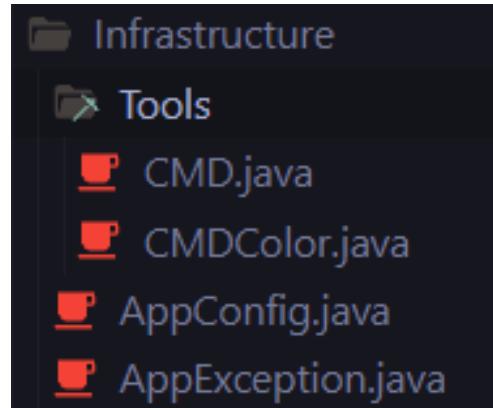


Figura 5.4: Directorio de Infrastructure

- **Directorio UserInterface**

Se encarga de almacenar las clases dirigidas a la presentación del usuario, recursos visuales, utilidades para personalización y pantallas del sistema.



Figura 5.5: Directorio de UserInterface

- **Directorio Storage**

Almacena las bases de datos creadas, los scripts utilizados para la gestión de la BD y el registro de errores o logs generados en el programa.

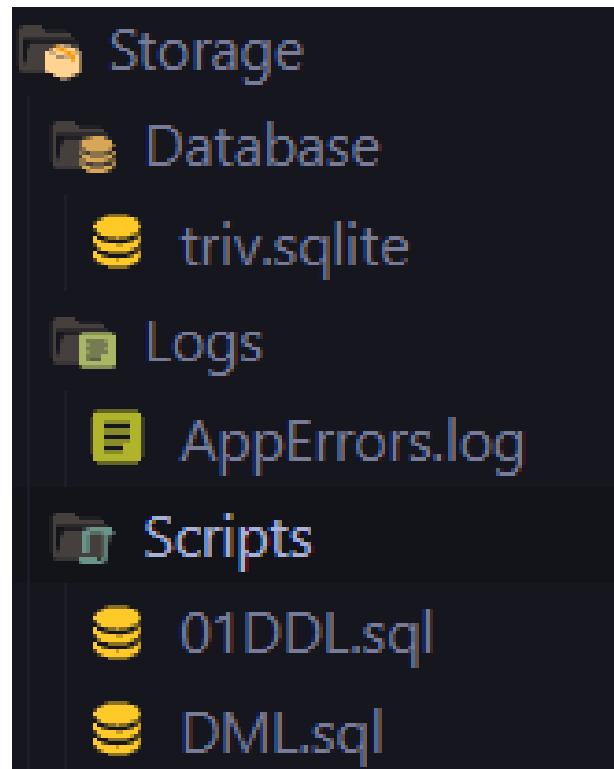


Figura 5.6: Directorio de Storage

# Capítulo 6

## Base de Datos

### 6.1. Sistema Gestor de Base de Datos (SGDB)

SQLite es un gestor de base de datos ligero y embebido, es decir, que se ejecuta internamente en la aplicación sin intervención de servicios externos, sino que funciona como una librería y, en este caso, una extensión de VSCode, la cual permite gestionar las acciones realizadas con la información almacenada en la Base de Datos mediante scripts de tipo sql.



Figura 6.1: Gestor de base de datos SQLite

## 6.2. Modelo Entidad Relación (MER)

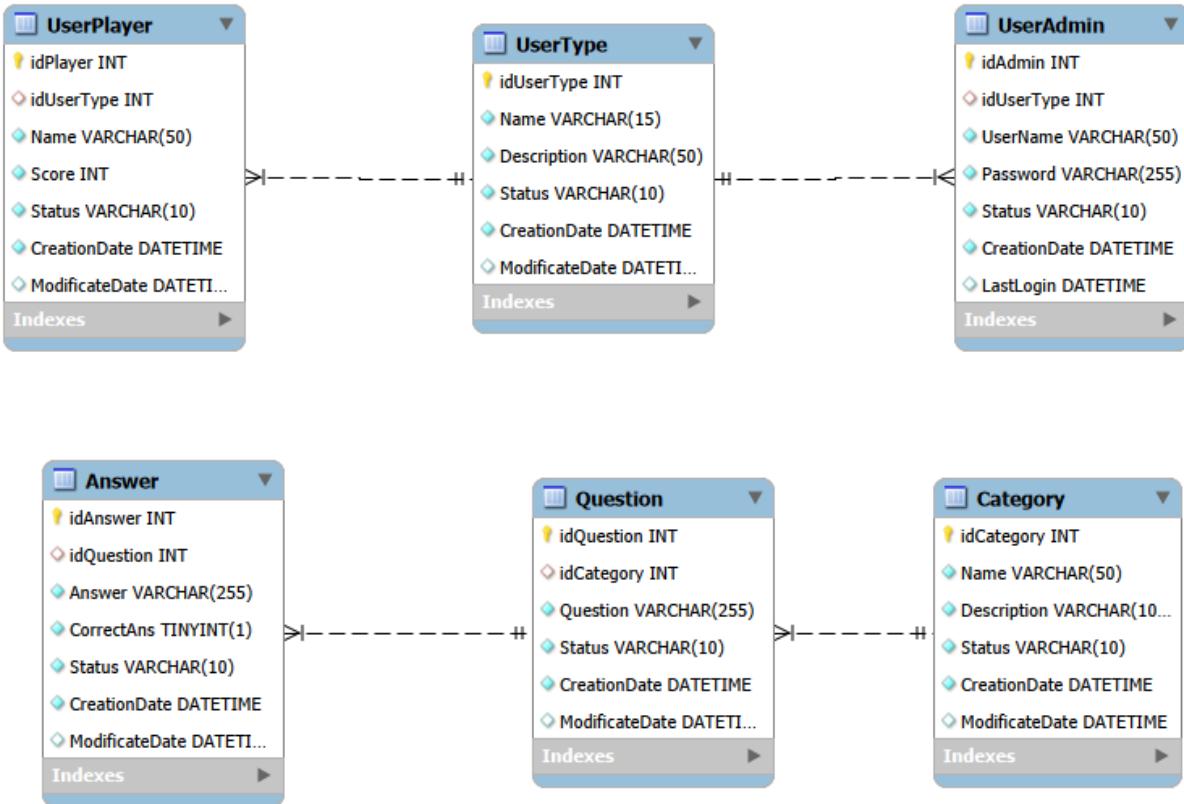


Figura 6.2: Modelo Entidad Relación del sistema

# Capítulo 7

## Lógica del Sistema

Describe los procesos principales del juego.

### 7.1. DataHelperSQLite

La clase DataHelperSQLite actúa como el componente común de infraestructura que heredan todas las clases DAO de la aplicación. Es una clase abstracta diseñada para agrupar la gestión de la conexión a la base de datos SQLite y proveer de información común, como el manejo de fechas a todas las clases DAO que hereden de ella y que están contenidas en la capa Data Access. La aplicación utiliza JDBC (Java Database Connectivity) como estándar para interactuar con la base de datos. Esto significa que DataHelperSQLite no se comunica directamente con el archivo de base de datos, sino que delega esta tarea al DriverManager de Java, cumpliendo los principios de mantenibilidad y flexibilidad. Al inicializar la clase, la variable DBPathConnection obtiene su valor llamando al método getDATABASE(), propio de la clase AppConfig. De esta manera, la conexión se establece mediante un path específico definido en src/app.properties bajo la clave db.File. El valor configurado es: `jdbc:sqlite:Storage\Database\triv.sqlite`, donde jdbc: Indica el protocolo principal, es decir, la API estándar de Java. Sqlite: Se encarga de asignar al DriverManager la tarea de buscar y cargar el driver específico de SQLite. En este caso, la librería sqlite-jdbc importada en el proyecto.

`Storage\Database\triv.sqlite`: Es la ruta relativa al recurso.

El método openConnection() se encarga de:

Invocar a DBPathConnection para recuperar la cadena JDBC externa.

Llamar a DriverManager.getConnection(), encargada de recorrer los drivers registrados en el CLASSPATH. Al detectar el prefijo sqlite:, selecciona la clase org.sqlite.JDBC.

Retornar un objeto `java.sql.Connection` activo, que se almacena en una variable estática para ser reutilizada por todos los DAOs, únicamente si la conexión es correcta.

Nótese que `openConnection()`, está declarado como `synchronized`, asegurando que, si dos hilos intentan conectar al mismo tiempo, no se creen dos conexiones diferentes, evitando conflictos de acceso a la base de datos.

## 7.2. Interfaz del Videojuego

Para el diseño del videojuego *Liminalis* se ha utilizado el método `SingleFrame`, el cual permite trabajar con un único `JFrame` durante toda la ejecución del juego. Este enfoque ayuda a mantener el tamaño del frame de forma estable, evitando cambios inesperados o la creación de nuevas ventanas. En lugar de crear múltiples frames, se reutiliza el mismo, repintando su contenido según la pantalla que se necesite mostrar.

## 7.3. Métodos de interfaz

Este método se implementó con el objetivo de *cambiar únicamente el panel dentro del frame*, permitiendo la transición entre distintas vistas sin necesidad de crear un nuevo `JFrame`. El método recibe como parámetro un nuevo `JPanel`, el cual se dibuja sobre el panel anterior mediante el proceso de repintado.

Esto permite:

- Un cambio fluido entre pantallas
- Mejor control de la interfaz
- Mayor eficiencia en el uso de recursos

## 7.4. Métodos de interfaz que regresan los páneles

Cada pantalla del juego corresponde a un *nuevo panel (JPanel)*, el cual se muestra al usuario mediante el repintado del frame. Estos métodos están organizados en *clases distintas*, lo que mejora la estructura del código y facilita su mantenimiento.

Este enfoque aporta:

- Mejor organización del proyecto

- Separación clara de responsabilidades
- Mayor facilidad para futuras modificaciones

## 7.5. Uso de layouts

Para una mejor organización visual de los componentes, se utilizaron distintos *layouts*, entre ellos:

- BorderLayout
- FlowLayout
- BoxLayout
- Otros según las necesidades de cada panel

El uso adecuado de layouts permitió una distribución clara, adaptable y ordenada de los elementos dentro de cada pantalla del juego.

## 7.6. ScreenKeyboard/Teclado en pantalla

El panel *ScreenKeyboard* es uno de los más importantes del juego, ya que permite al usuario *ingresar texto* en diferentes pantallas. Esta clase recibe como parámetro los campos de texto donde se realizará la escritura.

Entre sus principales funciones se incluyen:

- Cambio entre *mayúsculas y minúsculas*
- Borrado de texto
- Inserción de espacios
- Ingreso de números

Gracias a este panel, la interacción del usuario con el juego es más intuitiva y controlada.

## 7.7. ScreenGame/Pantalla de Juego

En el panel *ScreenGame* se presenta:

- La categoría de la pregunta
- El enunciado de la pregunta
- Cuatro opciones de respuesta

Cuando el usuario ingresa una respuesta correcta o válida, la pregunta y sus opciones se actualizan dinámicamente dentro del mismo panel, utilizando los métodos de *repintado* y *revalidación*.

Esta lógica se mantiene durante todo el desarrollo del juego, hasta completar el banco total de preguntas disponibles.

## 7.8. Relación de tablas

Cada tabla tiene su relación según el tratado de datos que necesita el sistema, en este caso, la tabla UserType tiene relación uno a muchos con la tabla UserAdmin y UserPlayer, dado que un tipo de usuario admin puede ser de muchos usuarios administradores, y un tipo de usuario jugador, puede ser de muchos jugadores. En adición con las relaciones para las preguntas, existe una relación uno a muchos entre las tablas Question y Answer, dado que una pregunta puede tener muchas respuestas (sean correctas o no) en las pantallas de presentación, mientras que la tabla Category con Question es de uno a muchos, ya que una categoria puede ser de muchas preguntas. Las relación se observa en el MER como se observa en la Figura 6.2

# Capítulo 8

## Descripción de Clases

A continuación, se presentan las clases más relevantes relacionadas con el dominio del sistema.

### ■ Clase AnswerBL

Posee los métodos encargados de leer las opciones de respuesta que serán abstraídas de la tabla Answer y la respuesta correcta correspondiente a la pregunta.

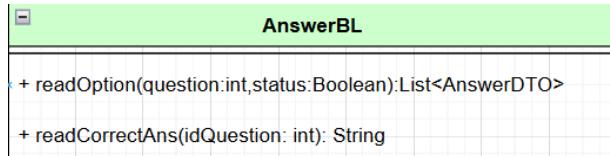


Figura 8.1: Diagrama de la clase AnswerBL

### ■ Clase QuestionBL

Se encarga de las acciones de leer las preguntas por id, traer todas las preguntas existentes, tomar la pregunta que se presentará en el juego y verificar si la pregunta tomada ya está en la lista de preguntas.

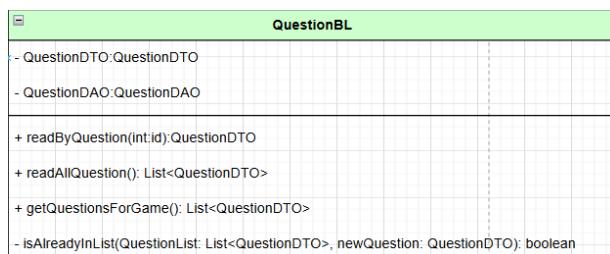


Figura 8.2: Diagrama de la clase QuestionBL

### ■ Clase UserAdminBL

Su función es garantizar las tareas de administrador y su autenticación, considerando también las acciones aplicadas a cada jugador: actualizar sus datos, cambiar el estado, leer sus estados, obtener las filas de la tabla y buscar al jugador por su nombre.

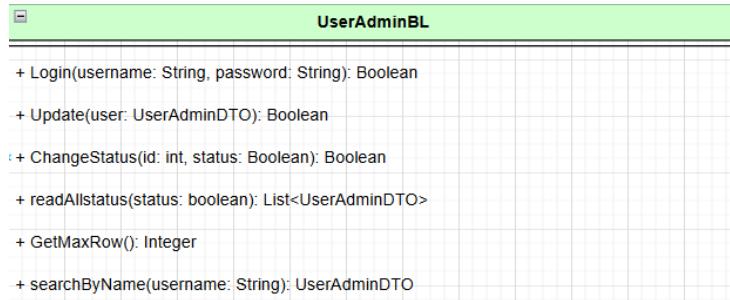


Figura 8.3: Diagrama de la clase UserAdminBL

### ■ Clase UserPlayerBL

Permite realizar acciones teniendo en cuenta al jugador: crear uno nuevo, verificar si ya existe, traer el id del jugador por nombre, mostrar todos los jugadores activos, actualizar los registros del jugador, actualizar registros por nombre de jugador, buscar por nombre, leer por id y obtener todos los jugadores.

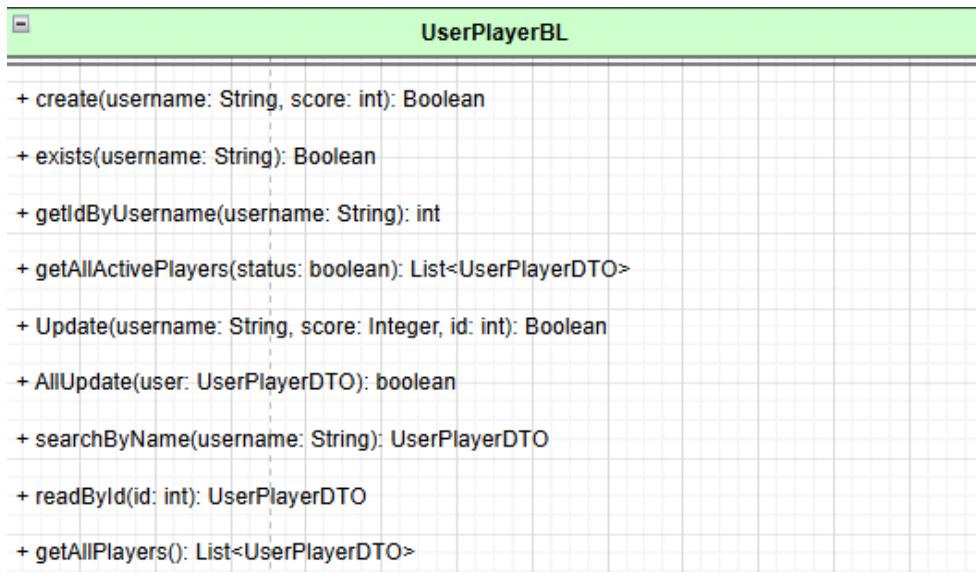


Figura 8.4: Diagrama de la clase UserPlayerBL

- Clase UserTypeBL

Garantiza las acciones de traer todos los registros de los tipos de usuario con su descripción, crear otro tipo de usuario en caso de existir, actualizar los registros y obtener las filas máximas de la tabla.

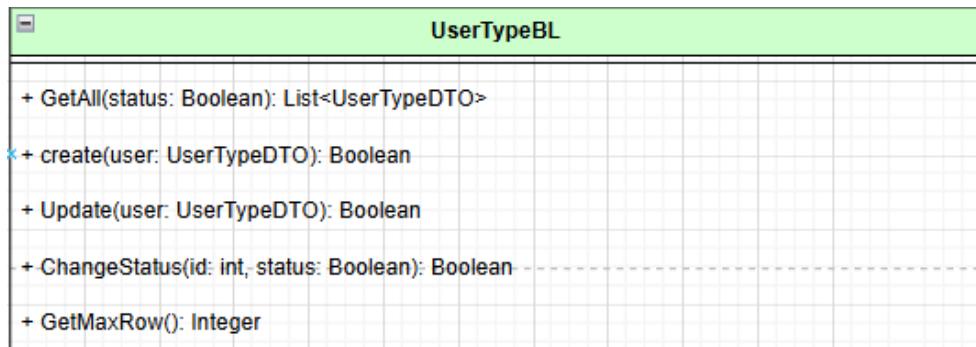


Figura 8.5: Diagrama de la clase UserTypeBL

# Capítulo 9

## Manejo de Errores

### 9.1. Arquitectura del Manejo de Errores

El manejo de errores se implementa de forma transversal en toda la aplicación, siguiendo una arquitectura por capas que permite capturar, propagar y presentar las excepciones de manera controlada.

Interfaz de Usuario (UI)



Lógica de Negocio (BL)



Acceso a Datos (DAO)



Sistema de Logs

Cada capa cumple una función específica dentro del proceso:

- **DAO:** Detecta y captura errores técnicos, principalmente relacionados con el acceso a la base de datos.
- **BL:** Reenvía las excepciones o añade contexto funcional cuando es necesario.
- **UI:** Presenta mensajes claros y controlados al usuario final.
- **Logs:** Registra de forma persistente toda la información del error.

## 9.2. Clase Base de Excepciones

Para unificar el tratamiento de errores se implementa la clase personalizada `AppException`, la cual extiende la clase `Exception` de Java. Esta clase encapsula tanto el mensaje mostrado al usuario como el contexto técnico del error.

### 9.2.1. Tipos de Excepciones

#### Excepciones Lógicas

Se utilizan para validaciones internas del sistema cuando no existe una excepción técnica subyacente.

```
[language=Java] throw new AppException( "El nombre del jugador no puede estar vacío");
```

#### Excepciones Técnicas

Se generan al capturar excepciones propias del sistema o de bibliotecas externas, proporcionando información adicional sobre el origen del error.

```
[language=Java] throw new AppException( "No se pudo crear el jugador: - entity.getName(), e, UserPlayerDAO.class, create");
```

## 9.3. Sistema de Registro de Errores

Toda excepción gestionada mediante `AppException` se registra automáticamente en un archivo de log, permitiendo el análisis posterior de fallos y auditoría del sistema.

### 9.3.1. Ubicación de los Archivos de Log

`%APPDATA%\Liminalis\Logs\errors.log`

### 9.3.2. Formato del Registro

```
SHOW  No se pudo crear el jugador: Juan
LOG   2026-01-27 14:35:22 |
UserPlayerDAO.create |
Duplicate entry 'Juan'
```

## 9.4. Propagación de Excepciones

El sistema utiliza un mecanismo de propagación de excepciones (burbujeo), en el cual los errores ascienden desde las capas inferiores hasta la interfaz de usuario.

1. El error se origina en la capa DAO.
2. Se encapsula en una instancia de `AppException`.
3. La excepción es reenviada por la capa de Lógica de Negocio.
4. La interfaz captura la excepción y muestra el mensaje correspondiente.
5. El evento queda registrado en el archivo de log.

## 9.5. Validaciones

El sistema implementa validaciones lógicas para prevenir errores antes de que ocurran fallos técnicos, entre ellas:

- Verificación de campos obligatorios.
- Control de duplicidad de registros.
- Validación del estado de la aplicación.

Estas validaciones generan excepciones controladas que informan claramente al usuario.

## 9.6. Buenas Prácticas

Para mantener la coherencia del manejo de errores se siguen las siguientes prácticas:

- Uso de mensajes descriptivos y consistentes.
- Inclusión del contexto de ejecución (clase y método).
- Reenvío de excepciones en la capa de negocio.
- Separación entre errores lógicos y técnicos.
- Revisión periódica de los archivos de log.

# Capítulo 10

## Instalación y Configuración



Figura 10.1: Aplicación Wix Toolset

Para la ejecucion del codigo en modo desarrollador, se utilizo el IDE **Visual Studio Code (VSCode)** junto con el **JDK de Java 25**, haciendo uso de la Maquina Virtual de Java (JVM). Sin embargo, para la creacion del instalador del videojuego, se empleo una herramienta incluida por defecto en el JDK llamada *jpackage*, la cual permite generar instaladores nativos para el sistema operativo. El proceso de compilacion e instalacion se realizo de la siguiente manera:

1. **Generacion del archivo .jar:** Se crea un archivo .jar del proyecto completo utilizando la herramienta *Java Projects* de VSCode. Este archivo contiene toda la

lógica y recursos necesarios del juego.

2. **Uso de WiX Toolset:** Se utiliza *WiX Tools* para leer el archivo .jar y convertirlo en un archivo .msi, permitiendo obtener un instalador nativo para Windows.
3. **Ejecucion de comandos con jpackage y WiX:** Mediante una linea de comandos, se integran *jpackage* y *WiX Tools* para empaquetar correctamente la aplicación.
4. **Creacion del instalador final:** Como resultado, se obtiene un instalador completo de la aplicación, el cual solo necesita ejecutarse para instalar el juego, incluyendo automáticamente la Maquina Virtual de Java, sin requerir que el usuario la tenga instalada previamente.