Documentación del Proyecto – SteamDB

1. Introducción

Este proyecto consiste en una aplicación llamada SteamDB, que simula una plataforma de distribución digital de videojuegos. A través de ella, los usuarios pueden ver juegos, comprar, escribir reseñas, y si son administradores, gestionar otros usuarios.

2. Motivación / Justificación de la idea

La motivación principal de este proyecto fue crear una réplica simplificada del sistema de gestión de una tienda digital como Steam, usando los conceptos vistos durante el curso.

Durante la realización del proyecto he realizado:

- Manejo de interfaces gráficas.
- Acceso a datos.
- Control de flujo entre pantallas.
- Validación y gestión de usuarios.

3. Objetivos propuestos

Los objetivos propuestos antes durante la realización del proyecto son los siguientes:

- Crear una interfaz gráfica con Java Swing.
- Implementar funcionalidades: login, registro, visualización de juegos, compras y reseñas.
- Diferenciar usuarios normales de administradores.
- Permitir que los administradores gestionen usuarios (eliminar).

4. Metodología utilizada

La metodología ágil que más se asemeja a la que he tenido durante la realización del proyecto sería Crystal.

En qué consiste: Se adapta al tamaño del equipo y al nivel de criticidad del proyecto. Es una de las metodologías más ligeras y flexibles.

Ideal para: Equipos pequeños que quieren libertad para organizarse según sus necesidades. Ya que fui adaptando el desarrollo a medida que avanzaba, sin seguir reglas estrictas pero asegurándome de entregar algo funcional.

5. Diagrama de Gantt

No realizado el diagrama

6. Tecnologías y herramientas utilizadas

Lenguaje: Java

Entorno de desarrollo: IntelliJ IDEA

GUI: Java Swing

Control de versiones: Git

Estilo visual: Personalizado con la clase cread “Estilos” inspirado en colores de Steam

Gestión de datos: Simulada mediante clases Java (Usuario, Juego, Reseña, Compra)

7. Análisis

Requisitos Funcionales

- RF1: El usuario puede iniciar sesión mediante su email.
- RF2: El sistema distingue entre usuarios normales y administradores.
- RF3: Los usuarios pueden ver juegos y sus reseñas.
- RF4: Los usuarios pueden realizar compras.
- RF5: Los administradores pueden eliminar usuarios.

Requisitos No Funcionales

- RNF1: El sistema debe validar el email del usuario.

Diagrama Entidad-Relación

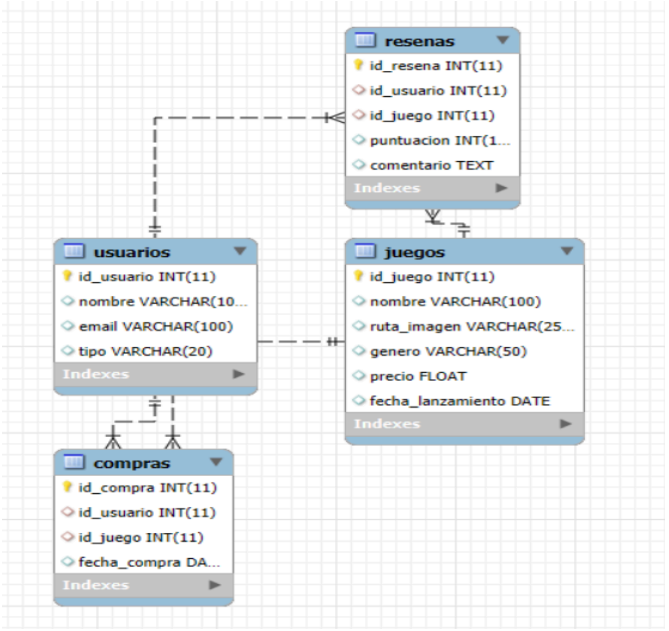


Diagrama de clases (También hay un archivo llamado diagramas.md)

```
%% ===== Paquete: tablas =====
class Usuario {
    - int id
    - String nombre
    - String email
    - String tipo
    + int getId()
    + String getNombre()
    + String getEmail()
    + String getTipo()
}

class Juego {
    - int idJuego
    - String nombre
    - String genero
    - float precio
    - String fechaLanzamiento
    + int getIdJuego()
    + String getNombre()
    + String getGenero()
    + float getPrecio()
    + String getFechaLanzamiento()
}

class Compra {
    - int id
    - int idUsuario
    - int idJuego
    - String fechaCompra
    + int getId()
    + String getIdUsuario()
    + String getIdJuego()
    + String getFechaLanzamiento()
}

class Resena {
    - int id
    - int idUsuario
    - int idJuego
    - int puntuacion
    - String comentario
    + int getId()
    + int getIdUsuario()
    + int getIdJuego()
    + int getPuntucacion()
    + String getComentario
}
```

```
%% ===== Paquete: gestores =====
class GestorUsuarios {
    + void insertarUsuario(Usuario usuario)
    + List<Usuario> obtenerTodosLosUsuarios()
    + Usuario buscarPorEmail(String email)
    + void borrarUsuario(int idUsuario, Usuario solicitante)
    + void registrarUsuario(Usuario usuario)
}

class GestorJuegos {
    + List<Juego> obtenerTodosLosJuegos()
    + void insertarJuego(Juego juego)
    + Juego obtenerPorId(int idJuego)
}

class GestorCompras {
    + void registrarCompra(Compra compra)
    + List<Compra> obtenerComprasPorUsuario(int idUsuario)
    + float calcularGastoTotal(int idUsuario)
    + List<String[]> obtenerComprasDeUsuarioConNombres(int idUsuario)
}

class GestorResenas {
    + void insertarResena(Resena resena)
    + List<Resena> obtenerResenasPorJuego(int idJuego)
    + void borrarResena(int idResena, int idUsuario)
    + List<String[]> obtenerResenasConNombres(int idJuego)
    + List<String[]> obtenerResenasDeUsuario(int idUsuario)
    + boolean editarResena(int usuarioId, String nombreJuego, String nuevoComentario, int nuevaPuntuacion)
}
```

```
mermaid
%% ===== Paquete: vistas =====
class VentanaLogin {
    - JTextField campoEmail
    - JButton botonEntrar
    - JButton botonCrearUsuario
}

class VentanaMenu {
    - Usuario usuario
}

class VentanaAdministrarUsuarios {
    - JTable tablaUsuarios
    - DefaultTableModel modelo
    - GestorUsuarios gestorUsuario
    - Usuario admin
}

class VentanaCompras {
    - JTable tabla;
    - DefaultTableModel modelo;
    - Usuario usuario;
    - GestorCompras gestorCompras
}

class VentanaCrearUsuario {
    - JTextField campoNombre;
    - JTextField campoEmail;
    - JComboBox<String> comboTipo;
    - JButton botonRegistrar;
}

class VentanaJuegos {
    - JTable tablaJuegos;
    - DefaultTableModel modelo;
    - Usuario usuario;
    - GestorJuegos gestorJuegos
    - GestorCompras gestorCompras
}

class VentanaMisResenas {
    - JTable tablaResenas;
    - DefaultTableModel modelo;
    - Usuario usuario;
    - GestorResenas gestorResenas
}

class VentanaResenas {
    - JTable tablaResenas;
    - DefaultTableModel modelo;
    - Usuario usuario;
    - GestorResenas gestorResenas
    - GestorJuegos gestorJuegos
}
```

```
%% ===== Relaciones de entidades =====
Usuario --> Compra : "1..*"
Usuario --> Resena : "1..*"
Juego --> Compra : "1..*"
Juego --> Resena : "1..*"

%% ===== Relaciones con gestores =====
GestorUsuarios --> Usuario
GestorJuegos --> Juego
GestorCompras --> Compra
GestorCompras --> Juego
GestorCompras --> Usuario
GestorResenas --> Resena
GestorResenas --> Usuario
GestorResenas --> Juego

%% ===== Relaciones entre vistas y modelos =====
VentanaLogin --> GestorUsuarios
VentanaLogin --> Usuario
VentanaLogin --> VentanaMenu

VentanaMenu --> Usuario
VentanaMenu --> VentanaJuegos
VentanaMenu --> VentanaCompras
VentanaMenu --> VentanaMisResenas
VentanaMenu --> VentanaAdministrarUsuarios

VentanaAdministrarUsuarios --> GestorUsuarios
VentanaAdministrarUsuarios --> Usuario

VentanaCrearUsuario --> GestorUsuarios

VentanaCompras --> GestorCompras
VentanaCompras --> Usuario

VentanaJuegos --> GestorJuegos
VentanaJuegos --> GestorCompras
VentanaJuegos --> Usuario

VentanaMisResenas --> GestorResenas
VentanaMisResenas --> Usuario

VentanaResenas --> GestorResenas
VentanaResenas --> GestorJuegos
VentanaResenas --> Usuario
```

Casos de uso

Caso de Uso 1	
Alias	CU1_Iniciar_Sesion
Actores	Usuario
Requisito funcional	RF-001
Descripción	El usuario puede iniciar sesión mediante su email.
Referencias	Ninguna
Comentarios	Ninguno

Caso de Uso 2	
Alias	CU2_Ver_Juegos
Actores	Usuario
Requisito funcional	RF-003
Descripción	Los usuarios pueden ver juegos y sus reseñas.
Referencias	RF-001
Comentarios	Ninguno

Caso de Uso 3	
Alias	CU3_Ver_Reseñas
Actores	Usuario
Requisito funcional	RF-003
Descripción	Usuario visualiza sus propias reseñas.
Referencias	RF-001
Comentarios	Ninguno

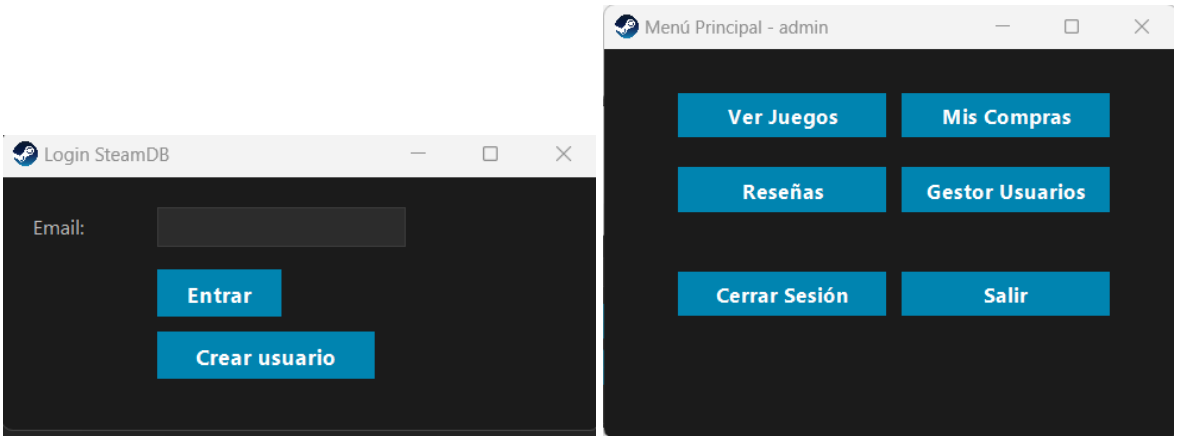
Caso de Uso 4	
Alias	CU4_Realizar_Compras
Actores	Usuario
Requisito funcional	RF-004
Descripción	Los usuarios pueden realizar compras
Referencias	RF-001
Comentarios	Ninguno

Caso de Uso 5	
Alias	CU5_Eliminar_Usuarios
Actores	Usuario
Requisito funcional	RF-005
Descripción	Los administradores pueden eliminar usuarios.
Referencias	RF-001
Comentarios	Ninguno

8. Diseño

Mock-up: No se realizó

Resultado final



9. Partes resaltables del código

Uso de dispose()

Esto permite que la aplicación funcione en una sola ventana, haciendo más limpia la experiencia de usuario. Esto lo comento porque previamente al inicio del proyecto cada vez que hacía click en un botón que tuviera que llevarme a otra ventana, por ejemplo, cuando pulsamos “Ver Juegos” se creaba una ventana nueva, por lo que al terminar de usar toda la aplicación acabas teniendo muchas ventanas abiertas.

Por ejemplo aquí, estamos en la VentanaMenu, donde tenemos acceso a todos los botones de la app, “Ver Juegos”, “Reseñas”, etc. Y lo que hacemos es poner un dispose() para cerrar la ventana y posteriormente new VentanaMisResenas(usuario) para que me mande a la siguiente ventana y así no tener siempre muchas abiertas.

```
// Botón para ver las reseñas que ha escrito en cualquier juego el usuario
JButton btnResenas = new JButton( text: "Reseñas");
btnResenas.setBounds( x: 50,  y: 80,  width: 140,  height: 30);
Estilos.estilizarBoton(btnResenas);
add(btnResenas);
btnResenas.addActionListener( ActionEvent e -> {
    dispose();
    new VentanaMisResenas(usuario);
});
```

Tabla con botón funcional en cada fila (JTable)

En la ventana de administración de usuarios, etc., se renderiza un botón "Eliminar" en cada fila para poder eliminar un usuario. También el propio uso de tablas es algo que he tenido que buscar como se hacía para cumplir con mi idea de hacer un listado de juegos, reseñas o usuarios.

```
/*
 JTable es el componente visual que representa una tabla.
 DefaultTableModel es el modelo de datos que contiene las filas y columnas.
 Aquí definimos las columnas: ID, Nombre, Correo, Tipo (admin o no), y un botón de acción.
*/
modelo = new DefaultTableModel(new Object[]{"ID", "Nombre", "Correo", "Tipo", "Acción"}, rowCount: 0);
tablaUsuarios = new JTable(modelo);
JScrollPane scroll = new JScrollPane(tablaUsuarios); // JScrollPane permite hacer scroll si hay muchos usuarios
scroll.setBounds( x: 20, y: 40, width: 540, height: 200);
add(scroll);

JButton btnVolver = new JButton( text: "Volver");
btnVolver.setBounds( x: 450, y: 260, width: 100, height: 30);
Estilos.estilizarBoton(btnVolver);
add(btnVolver);

cargarUsuarios();
```

```
// Clase para personalizar el renderizador de botón
// Esta clase hace que la celda de la columna "Acción" se vea como un botón.
class ButtonRenderer extends JButton implements TableCellRenderer {  1 usage  Jacobo-Dominguez
    public ButtonRenderer() { setOpaque(true); }

    @Override  Jacobo-Dominguez
    public Component getTableCellRendererComponent(JTable table, Object value, boolean isSelected,
                                                   boolean hasFocus, int row, int column) {
        setText("Eliminar");
        return this;
    }
}
```

```
// Esta clase hace que el botón funcione: cuando haces clic, elimina al usuario correspondiente.
class ButtonEditor extends DefaultCellEditor {  1 usage  Jacobo-Dominguez
    private String label;  3 usages
    private int usuarioId;  2 usages

    public ButtonEditor(JCheckBox checkBox) { super(checkBox); }

    @Override  Jacobo-Dominguez
    public Component getTableCellEditorComponent(JTable table, Object value, boolean isSelected, int row, int column) {
        label = (value == null) ? "" : value.toString();
        usuarioId = (int) table.getValueAt(row,  column: 0);  // Obtener el ID del usuario
        JButton btnEliminar = new JButton(label);
        Estilos.estilizarBoton(btnEliminar);
        btnEliminar.addActionListener(new ActionListener() {  Jacobo-Dominguez
            @Override  Jacobo-Dominguez
            public void actionPerformed(ActionEvent e) {
                eliminarUsuario(usuarioId);  // Eliminar el usuario usando el metodo que ya definiste
            }
        });
        return btnEliminar;
    }

    @Override  Jacobo-Dominguez
    public Object getCellEditorValue() { return label; }
}
```

10. Conclusión del trabajo y del curso

Este proyecto me ha permitió aplicar de forma práctica todo lo aprendido en la asignatura de Programación y Base de Datos:

Durante el desarrollo, he utilizado varios conceptos de programación orientada a objetos (POO), como:

- Clases, objetos.
- Uso de JFrame, JTable, JButton, JTextField, JComboBox, etc. para interfaces gráficas en Java Swing.
- Implementación de listeners y eventos para responder a acciones del usuario.

Desde el punto de vista de base de datos, se han aplicado los siguientes conocimientos:

- Diseño de una base de datos relacional, con tablas como Usuario, Juego, Compra y Reseña.
- Identificación de relaciones entre tablas
- Creación de un diagrama entidad-relación (ER)
- Uso de consultas SQL para recuperar, insertar, actualizar o eliminar datos.
- Aprender a cómo se conectan los datos de base de datos a un IDE como Eclipse o Intellij