Lexis MANAGE

Rojas Herreño Cristian Alexis
cristian.rojas.herreno@pi.edu.co
Politécnico Internacional

Resumen – Este documento presenta **Lexis Manage**, un sistema de información desarrollado para gimnasios, academias y otros establecimientos que gestionan membresías. El sistema permite una administración eficiente de datos de usuarios, membresías y pagos, optimizando el manejo de la información. Además, cuenta con una interfaz gráfica intuitiva, diseñada para que cualquier usuario, sin importar su nivel técnico, pueda utilizarla fácilmente. Lexis Manage fue desarrollado en **Java** utilizando **NetBeans**, con una base de datos alojada en un servidor **Ubuntu** dentro de una **máquina virtual** en **VirtualBox**. A lo largo del documento, se detallan sus funcionalidades clave, la arquitectura técnica y los requisitos para su implementación.

Palabras Clave: Gestión de membresías, Sistema de, información, Interfaz gráfica intuitiva, Java, NetBeans, Servidor Ubuntu, Base de datos, VirtualBox, Administración de usuarios, Optimización de datos

Abstract -This document presents **Lexis Manage**, an information system designed for establishments such as gyms, academies, and other membership-based businesses. The system enables efficient management of user data, memberships, and payments, optimizing information handling. Additionally, it features an intuitive graphical interface, allowing users of any technical level to navigate it easily. Lexis Manage was developed in **Java** using **NetBeans** and connects to a database hosted on an **Ubuntu server** within a **VirtualBox** virtual machine. This document details the system's key functionalities, technical architecture, and implementation requirements.

Keywords: Membership Management, Information System, Intuitive Graphical Interface, Java, NetBeans, Ubuntu Server, Database, VirtualBox, User Administration, Data Optimization.

I. INTRODUCCION

En un entorno donde la digitalización y la gestión eficiente de datos se vuelven esenciales para el éxito de cualquier negocio, la necesidad de un sistema de información robusto es particularmente relevante para pequeñas academias y gimnasios que están comenzando o que aún operan con métodos manuales. Este proyecto, Lexis Manage, se ha desarrollado con el propósito de ofrecer una solución accesible y efectiva para estos establecimientos.

Muchos de estos negocios enfrentan desafíos significativos en la gestión de sus operaciones diarias, como la administración de membresías, el control de pagos y el seguimiento de datos de usuarios. Sin un sistema organizado, la información se dispersa y se vuelve difícil de manejar, lo que puede llevar a errores y a una mala experiencia del cliente. Lexis Manage busca resolver estas problemáticas proporcionando un sistema estructurado que asegura la integridad de los datos y permite un control eficiente de la información de los usuarios.

El objetivo principal de este software es proporcionar un marco que permita a los administradores de gimnasios y academias mantener un control claro y ordenado sobre sus operaciones. Al implementar Lexis Manage, estos establecimientos no solo optimizarán su gestión, sino que también obtendrán una visión más clara de su información, lo que les permitirá tomar decisiones informadas y mejorar la experiencia de sus usuarios.

Este documento está estructurado de la siguiente manera: comenzaremos con un resumen del sistema y sus funcionalidades clave, seguido de un análisis detallado de la arquitectura técnica, las especificaciones del sistema y las diversas herramientas utilizadas en su desarrollo. Además, se incluirán diagramas que ilustran la estructura del software y su interacción con la base de datos, proporcionando una comprensión integral de Lexis Manage.

II. Metodología

El desarrollo de **Lexis Manage** se llevó a cabo utilizando **Java** y el entorno de desarrollo integrado **NetBeans**, que proporcionan un marco robusto y eficiente para la creación de aplicaciones de software. Para la gestión de datos, se implementó una conexión a una base de datos alojada en un **servidor Ubuntu**, configurado específicamente para almacenar y gestionar la información de los usuarios y las operaciones del sistema.

El proceso de conexión entre la aplicación y la base de datos se inicia en el momento en que el usuario ejecuta el programa. Esta conexión permite al sistema acceder a los datos necesarios para gestionar las membresías, realizar pagos y actualizar la información de los usuarios en tiempo real.

Durante la ejecución, la aplicación envía consultas SQL al servidor para interactuar con la base de datos, asegurando que toda la información se maneje de manera eficiente y segura.

Al cerrar la aplicación, la conexión a la base de datos se finaliza automáticamente, lo que garantiza que los recursos del servidor se gestionen de forma adecuada y se mantenga la seguridad de los datos. Este enfoque de conexión dinámica permite una administración fluida y efectiva de la información, facilitando la operatividad del sistema para los usuarios finales.

III. Teoría del color

La paleta de colores utilizada en **Lexis Manage** está compuesta por una selección de tonos que, además de ser estéticamente agradables, cumplen con principios de diseño minimalista. La combinación de colores elegida no solo se basa en una preferencia personal, sino que también busca mejorar la usabilidad y la experiencia general del usuario.

Rojo (#ff4d58): Este color vibrante se utiliza para llamar la atención sobre acciones importantes y notificaciones. El rojo evoca sentimientos de energía y urgencia, lo que lo hace ideal para botones de acción y alertas.

Blanco (#ffffff): El blanco se emplea principalmente en los textos, proporcionando claridad y facilitando la lectura. Este color simboliza simplicidad y limpieza, lo que permite que la información destaque sobre los fondos oscuros.

Gris claro (#abbdc9): Este tono suave actúa como un color neutro que complementa los colores más fuertes. Se utiliza en elementos secundarios y de fondo, aportando un sentido de equilibrio y modernidad.

Azul oscuro (#1a2737): Este color se utiliza como fondo en varias secciones de la interfaz, creando un contraste efectivo con el texto blanco. El azul oscuro proporciona una base profesional y elegante, asociada con la confianza y la estabilidad.

Negro (#0a101e): Este color se emplea en áreas de fondo y en elementos de diseño, ofreciendo profundidad y un sentido de sofisticación. El negro ayuda a resaltar el contenido en blanco y gris claro, reforzando el enfoque minimalista del diseño.

La flexibilidad del sistema permite la posibilidad de cambiar la paleta de colores en el futuro, ofreciendo la oportunidad de adaptarse a diferentes preferencias de los usuarios y tendencias de diseño.

IV. Diseño de Aplicación

A. Definición de Requerimientos Funcionales ,No Funcionales(Historias de Usuario).

Las historias de usuario son descripciones breves y simples de las funcionalidades que el sistema debe ofrecer desde la perspectiva de los usuarios finales. Estas historias ayudan a comprender las necesidades y expectativas de los diferentes tipos de usuarios, garantizando que el desarrollo del sistema se enfoque en proporcionar un valor real a los usuarios y facilitar sus tareas diarias. A continuación, se presentan las historias de usuario identificadas para el sistema Lexis Manage, clasificadas en funcionales y no funcionales.

ID	Tipo	Historia de Usuario
1	Funcional	Como administrador, quiero tener la capacidad de registrarme y acceder al sistema de manera segura para gestionar la información del establecimiento.
2	Funcional	Como administrador, quiero agregar, editar y visualizar la información de los usuarios del establecimiento para mantener un registro actualizado y accesible.
3	Funcional	Como administrador, deseo tener un apartado de CRUD para gestionar las membresías, donde pueda agregar, editar y eliminar la información de las membresías que el establecimiento ofrece.
4	Funcional	Como administrador, quiero establecer un control de acceso para que solo los usuarios con membresías activas puedan ingresar al establecimiento.
5	Funcional	Como administrador, quiero añadir pagos de los usuarios al adquirir o actualizar sus membresías para llevar un control adecuado de las transacciones.

6	Funcional	Como administrador, deseo generar reportes y estadísticas sobre el establecimiento, incluyendo la asistencia de usuarios e ingresos por pagos, para tener una visión clara del rendimiento del negocio.
7	Funcional	Como administrador, quiero manejar la información de los empleados del establecimiento y asignar roles específicos para optimizar la administración del personal.
8	Funcional	Como administrador, deseo acceder a un apartado de configuración donde pueda personalizar la información de la aplicación según las necesidades del establecimiento, mejorando así la experiencia del usuario.
9	No Funcional	Como usuario, quiero que el sistema responda rápidamente (menos de 2 segundos) a mis acciones para garantizar una experiencia fluida y eficiente.
10	No Funcional	Como usuario, deseo que la interfaz sea intuitiva y fácil de navegar, permitiendo a cualquier persona usar el sistema sin necesidad de una capacitación extensa.
11	No Funcional	Como administrador, quiero que el sistema implemente medidas de seguridad robustas, como la autenticación de dos factores, para proteger la información sensible de los usuarios y del establecimiento.
12	No Funcional	Como administrador, deseo que el sistema sea escalable, permitiendo la incorporación de más usuarios y funciones sin afectar el rendimiento.
13	No Funcional	Como administrador, deseo que el sistema tenga un mantenimiento fácil y eficiente para garantizar que se mantenga actualizado y funcione sin problemas.

B. Definición del diagrama de arquitectura de mi proyecto.

El sistema **Lexis Manage** se basa en una arquitectura cliente-servidor. El usuario interactúa con el software desarrollado en Java, que presenta una interfaz gráfica intuitiva. Dependiendo de las acciones que el usuario realice en la GUI, el software envía consultas al servidor, que está alojado en un Ubuntu Server ejecutándose en VirtualBox. La base de datos, que se utiliza para almacenar y gestionar la información del sistema, reside en este servidor. Esta interacción permite que el usuario reciba respuestas a sus peticiones de manera eficiente y que el software gestione las solicitudes del usuario de manera efectiva.

C. Diagrama de Clases

D. Diagrama de secuencia.

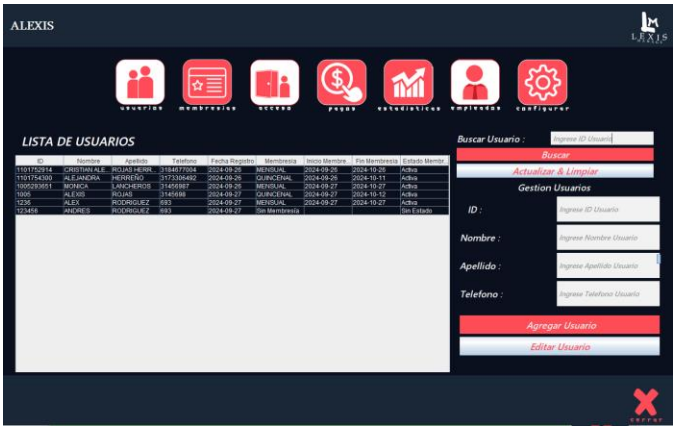
V. Mockups (Interfaz Gráfica).

1) GUI LOGIN.

2) GUI REGISTRO.

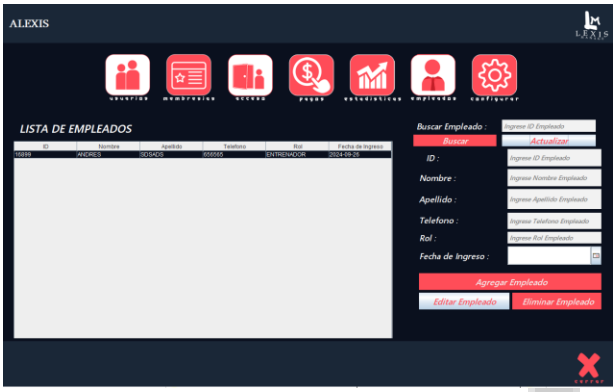
3) GUI OLVIDASTE.

4) GUI MENU PRINCIPAL-USUARIOS.

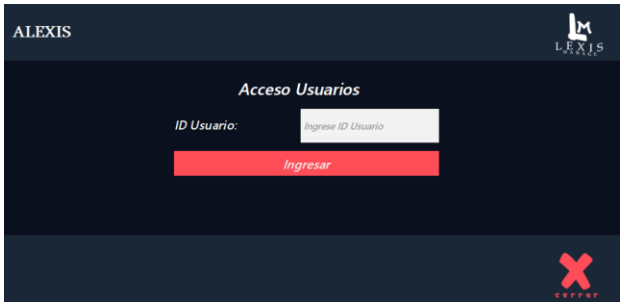


5) GUI MENU PRINCIPAL-MEMBRESIAS.

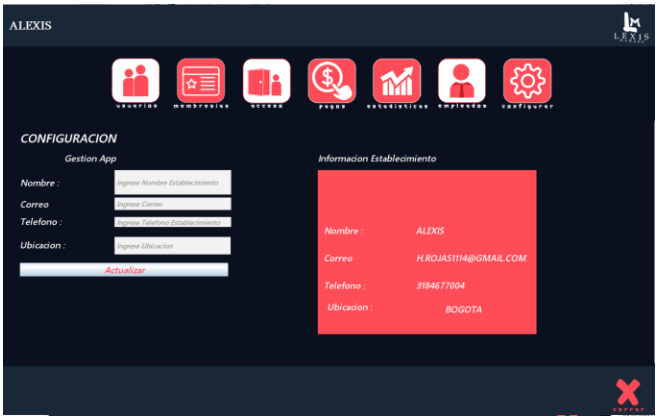
9) GUI MENU PRINCIPAL-EMPLEADOS.



6) GUI ACCESO USUARIOS.



10) GUI MENU PRINCIPAL-CONFIGURACION.



7) GUI MENU PRINCIPAL-PAGOS.

8) GUI MENU PRINCIPAL-ESTADISTICAS.

VI. Referencia del Servidor y Tecnologías.

1) Infraestructura Técnica

- **Servidor**: Se utiliza un servidor virtual con Ubuntu Server configurado en VirtualBox. Este servidor incluye la instalación de MySQL y la configuración necesaria para la gestión de usuarios y privilegios.
- **Puertos**: Se activan puertos específicos para permitir conexiones desde el programa y acceder al servidor de manera remota a través de un gestor de base de datos.

2) Base de Datos.

- **Gestor de Base de Datos:** Se utiliza MySQL como sistema de gestión de bases de datos.
- **Estructura de la Base de Datos:**

A continuación, se presentan las tablas principales que componen la base de datos, incluyendo las relaciones entre ellas:

```
CREATE TABLE USUARIO_ADMIN (
    id_usuario INT PRIMARY KEY AUTO_INCREMENT,
    nombre_usuario VARCHAR(50) NOT NULL,
    contrasena VARCHAR(255) NOT NULL
);

CREATE TABLE Preguntas_Seguridad (
    id_pregunta INT PRIMARY KEY AUTO_INCREMENT,
    pregunta VARCHAR(255) NOT NULL
);

CREATE TABLE Seguridad_UsuariosAdmin (
    id_seguridad INT PRIMARY KEY AUTO_INCREMENT,
    id_usuario INT,
    id_pregunta INT,
    respuesta_seguridad VARCHAR(255) NOT NULL,
    FOREIGN KEY (id_usuario) REFERENCES USUARIO_ADMIN(id_usuario),
    FOREIGN KEY (id_pregunta) REFERENCES Preguntas_Seguridad(id_pregunta)
);

CREATE TABLE Sesiones_Activas (
    id_sesion INT PRIMARY KEY AUTO_INCREMENT,
    id_usuario INT,
    fecha_inicio DATETIME,
    fecha_fin DATETIME,
    FOREIGN KEY (id_usuario) REFERENCES USUARIO_ADMIN(id_usuario)
);

CREATE TABLE MetodoPago (
    id_metodo_pago INT PRIMARY KEY AUTO_INCREMENT,
    metodo VARCHAR(50) NOT NULL
);

CREATE TABLE Membresia (
    id_membresia INT PRIMARY KEY AUTO_INCREMENT,
    nombre VARCHAR(100) NOT NULL,
    duracion INT NOT NULL,  -- En días
    informacion TEXT,
    costo DECIMAL(10, 2) NOT NULL
);

CREATE TABLE EstadoMembresia (
    id_estado INT PRIMARY KEY AUTO_INCREMENT,
    estado VARCHAR(50) NOT NULL
);

CREATE TABLE Usuario (
    id_usuario INT PRIMARY KEY AUTO_INCREMENT,
    identificacion VARCHAR(20) UNIQUE NOT NULL,
    nombre VARCHAR(100) NOT NULL,
    apellido VARCHAR(100) NOT NULL,
    telefono VARCHAR(20),
    fecha_ingreso DATE NOT NULL,
    id_estado INT,
    FOREIGN KEY (id_estado) REFERENCES EstadoMembresia(id_estado)
);

CREATE TABLE Pago (
    id_pago INT PRIMARY KEY AUTO_INCREMENT,
    id_usuario INT,
    id_membresia INT,
    fecha_inicio DATE NOT NULL,
    fecha_fin DATE NOT NULL,
    id_metodo_pago INT,
    FOREIGN KEY (id_usuario) REFERENCES Usuario(id_usuario),
    FOREIGN KEY (id_membresia) REFERENCES Membresia(id_membresia),
    FOREIGN KEY (id_metodo_pago) REFERENCES MetodoPago(id_metodo_pago)
);

CREATE TABLE Ingresos (
    id_ingreso INT PRIMARY KEY AUTO_INCREMENT,
    id_pago INT,
    fecha DATE NOT NULL,
    monto DECIMAL(10, 2) NOT NULL,
    FOREIGN KEY (id_pago) REFERENCES Pago(id_pago)
);

CREATE TABLE ResultadoAcceso (
    id_resultado INT PRIMARY KEY AUTO_INCREMENT,
    resultado VARCHAR(50) NOT NULL --
);

CREATE TABLE Acceso (
    id_acceso INT PRIMARY KEY AUTO_INCREMENT,
    id_usuario INT NOT NULL,
    fecha_acceso DATETIME NOT NULL,
    id_resultado INT NOT NULL,
    FOREIGN KEY (id_usuario) REFERENCES Usuario(id_usuario),
```

```
    FOREIGN KEY (id_resultado) REFERENCES ResultadoAcceso(id_resultado)
  );

  CREATE TABLE Personalizacion_Establecimiento (
    id_establecimiento INT PRIMARY KEY AUTO_INCREMENT,
    nombre VARCHAR(100) NOT NULL,
    correo VARCHAR(100),
    telefono VARCHAR(20),
    ubicacion VARCHAR(255),
    configurado BOOLEAN DEFAULT FALSE,
    id_admin INT NOT NULL,
    FOREIGN KEY (id_admin) REFERENCES USUARIO_ADMIN(id_usuario)
  );

  CREATE TABLE Empleados (
    id_empleado INT PRIMARY KEY AUTO_INCREMENT,
    identificacion VARCHAR(20) UNIQUE NOT NULL,
    nombre VARCHAR(100) NOT NULL,
    apellido VARCHAR(100) NOT NULL,
    telefono VARCHAR(20),
    rol VARCHAR(50) NOT NULL,
    fecha_ingreso DATE NOT NULL,
    id_admin INT NOT NULL,
    FOREIGN KEY (id_admin) REFERENCES USUARIO_ADMIN(id_usuario)
  );
```

B. Tecnologías Utilizadas

- **Lenguaje de Programación**: Java.
- **Entorno de Desarrollo**: NetBeans.
- **Bibliotecas Utilizadas**:

`java.util`: Para manejo de utilidades.
`javax.swing`: Para la creación de la interfaz gráfica.
`java.sql`: Para la conexión y manejo de la base de datos.

C. EDR (Diagrama de Entidad-Relación).

D. Relaciones entre las Tablas

1) **USUARIO_ADMIN**

- Relación: 1 a N con Seguridad_UsuariosAdmin (Un administrador puede tener múltiples preguntas de seguridad).
- Relación: 1 a N con Sesiones Activas (Un administrador puede tener múltiples sesiones activas).
- Relación: 1 a N con Personalizacion_Establecimiento (Un administrador puede personalizar múltiples establecimientos).
- Relación: 1 a N con Empleados (Un administrador puede gestionar múltiples empleados).

2) **Preguntas Seguridad**

- Relación: **1 a N** con `Seguridad_UsuariosAdmin` (Una pregunta de seguridad puede estar asociada a múltiples administradores).

3) **Seguridad_UsuariosAdmin**

- Relación: **N a 1** con `USUARIO_ADMIN` (Cada respuesta de seguridad pertenece a un único administrador).
- Relación: **N a 1** con `Preguntas_Seguridad` (Cada respuesta de seguridad está relacionada con una única pregunta).

4) **Sesiones_Activas**

- Relación: **N a 1** con `USUARIO_ADMIN` (Cada sesión activa pertenece a un único administrador).

5) **MetodoPago**

- Relación: **1 a N** con `Pago` (Un método de pago puede ser utilizado en múltiples pagos).

6) **Membresia**

- Relación: **1 a N** con `Pago` (Una membresía puede estar asociada a múltiples pagos).
- Relación: **1 a N** con `Usuario` (Una membresía puede ser asignada a múltiples usuarios).

7) **EstadoMembresia**

- Relación: **1 a N** con `Usuario` (Un estado de membresía puede estar asociado a múltiples usuarios).

8) **Usuario**

- Relación: **1 a N** con `Pago` (Un usuario puede tener múltiples pagos).
- Relación: **1 a N** con `Acceso` (Un usuario puede tener múltiples registros de acceso).
- Relación: **N a 1** con `Estado Membresía` (Cada usuario tiene un único estado de membresía).

9) **Pago**

- Relación: **N a 1** con `Usuario` (Cada pago pertenece a un único usuario).
- Relación: **N a 1** con `Membresia` (Cada pago está asociado a una única membresía).
- Relación: **N a 1** con `MetodoPago` (Cada pago utiliza un único método de pago).
- Relación: **1 a N** con `Ingresos` (Un pago puede generar múltiples ingresos).

10) **Ingresos**

- Relación: **N a 1** con `Pago` (Cada ingreso está asociado a un único pago).

11) **ResultadoAcceso**

- Relación: **1 a N** con `Acceso` (Un resultado de acceso puede estar asociado a múltiples registros de acceso).

12) **Acceso**

- Relación: **N a 1** con `Usuario` (Cada registro de acceso pertenece a un único usuario).
- Relación: **N a 1** con `ResultadoAcceso` (Cada registro de acceso tiene un resultado asociado).

13) **Personalizacion_Establecimiento**

- Relación: **N a 1** con `USUARIO_ADMIN` (Cada establecimiento personalizado pertenece a un único administrador).

14) **Empleados**

Relación: **N a 1** con `USUARIO_ADMIN` (Cada empleado es gestionado por un único administrador).

VII. Conclusiones

- **Integridad de Datos**: Las relaciones establecidas entre las tablas aseguran la integridad referencial, lo que significa que no se pueden crear registros en tablas secundarias sin que existan los registros correspondientes en las tablas primarias.
- **Facilidad de Mantenimiento**: Al tener un modelo bien estructurado, se facilita el mantenimiento y la escalabilidad del sistema, permitiendo futuras adiciones de funcionalidades sin afectar la estructura existente.
- **Seguridad**: Las tablas de seguridad y sesiones activas ayudan a proteger el sistema contra accesos no autorizados, asegurando que solo los administradores autenticados puedan gestionar la información del gimnasio.
- **Flexibilidad**: Las relaciones permiten que un usuario pueda tener múltiples membresías y pagos, ofreciendo flexibilidad en la gestión de membresías y permitiendo cambios según las necesidades del usuario.
- **Análisis de Datos**: La estructura de las tablas permite la generación de reportes y análisis de datos, facilitando la toma de decisiones informadas para la gestión del gimnasio.
- **Escalabilidad**: El diseño permite agregar más funcionalidades, como la incorporación de nuevos tipos de membresías o métodos de pago, sin necesidad de rediseñar la base de datos desde cero.

DOCUMENTO DESARROLLADO POR CRISTIAN ROJAS EN SEPTIEMBRE DE 2024.