Descripción breve

Sistema de gestión de licencias desarrollado con fines educativos para la asignatura de Base de datos, el programa simula las funcionalidades de un sistema de gestión de licencias real.

Documentación SGL

Autores

Angel A. Hernández Alonso

Kristian Águila Luna

Adrián Alonso Méndez

# Resumen

El *Sistema de Gestión de Licencias* es un proyecto educativo desarrollado para la asignatura de **Bases de Datos**, que simula las funcionalidades de un sistema real de emisión y seguimiento de licencias de conducción. Su propósito principal es facilitar el aprendizaje de:

* **Diseño y estructuración de bases de datos relacionales**.
* **Consultas SQL complejas** (JOINs, filtros, agregaciones)

**Tecnologías Utilizadas**:

* **Backend**: Java (lógica de negocio y conexión a BD).
* **Base de Datos**: PostgreSQL (modelado relacional, índices, validaciones).
* **Frontend**: JavaFX (interfaz gráfica intuitiva).

**Objetivos Clave**:

1. **Aprendizaje de SQL**:
   * Realizar consultas para gestionar conductores, licencias, infracciones y exámenes médicos.
   * Generar reportes (ej: licencias vencidas, infracciones por tipo).
2. **Diseño de BD**:
   * Modelar entidades como Conductor, Licencia, Examen e Infracción con relaciones (1:N, N:M).
   * Aplicar normalización y optimización (índices en campos frecuentes como CI o número de licencia).
3. **Arquitectura Multicapa**:
   * Separar claramente la interfaz (JavaFX), la lógica (Java) y los datos (PostgreSQL).

**Funcionalidades Destacadas**:

* **Roles y Permisos**:
  + *Administrador*, *Médicos* y *Trabajadores de Autoescuela* tienen accesos diferenciados.
* **Validaciones**:
  + Restricciones de integridad en PostgreSQL (ej: CHECK para fechas de vencimiento).
  + Validadores en Java (ej: formato de CI, campos obligatorios).
* **Reportes**:
  + Exportables a PDF/Excel (ej: listado de licencias emitidas en un período).

# Índice

Tabla de contenido

[Resumen 1](#_Toc199607995)

[Índice 2](#_Toc199607996)

[Requisitos 3](#_Toc199607997)

[Requisitos funcionales 3](#_Toc199607998)

[Requisitos no funcionales 3](#_Toc199607999)

[Mostrar reportes 5](#_Toc199608000)

[Solución propuesta 8](#_Toc199608001)

[Diagrama entidad relación 8](#_Toc199608002)

[Diagrama físico 8](#_Toc199608003)

[Entidades principales 9](#_Toc199608004)

[Relaciones 10](#_Toc199608005)

[Arquitectura 11](#_Toc199608006)

[Seguridad 12](#_Toc199608007)

[Roles 12](#_Toc199608008)

[Encriptación 12](#_Toc199608009)

[Interfaz Usuario 13](#_Toc199608010)

[Tratamiento de errores 18](#_Toc199608011)

# Requisitos

## Requisitos funcionales

Gestión Entidades (Administrador)

* Crear
* Modificar
* Agregar trabajador
* Filtrar por
  + Tipo (Clínica, Autoescuela)

Gestión de Conductores (Administrador)

* Crear
* Modificar
* Eliminar
* Ver todos

Gestión de Infracciones (Administrador)

* Registrar
* Ver Todas
* Filtrar
  + Tipo

Gestión de Licencias (Administrador)

* Crear
* Filtrar por
  + Categoría
  + Tipo
* Ver todas

Gestión de Exámenes (Administrador, Trabajador-Autoescuela, Medico)

* Registrar
* Ver todos
* Filtrar por
  + Tipo (Medico, Teórico, Practico)
  + Resultado (Aprobado, Reprobado)
  + Entidad (Clínica, Autoescuela)

## Requisitos no funcionales

Rendimiento

* **Tiempos de respuesta:**
  + Operaciones CRUD (crear, modificar, eliminar registros): **< 1 segundo** en condiciones normales.
  + Consultas complejas con JOINs (ej: reportes): **< 3 segundos** con hasta 50,000 registros.
* **Carga inicial:** El sistema debe cargar la interfaz principal en **< 2 segundos** al iniciar.

Seguridad

* **Autenticación:**
  + Contraseñas encriptadas.
* **Integridad de datos:**
  + Validación de tipos y restricciones a nivel de **PostgreSQL** (CHECK, NOT NULL, FK).

Usabilidad

* **Interfaz (JavaFX/FXML):**
  + Mensajes de error contextuales (ej: "La licencia ya existe" en lugar de "Error SQL 23505").
  + Interfaces intuitivas y fáciles de usar.

Compatibilidad

* **Entorno de ejecución:**
  + **Java 17** (LTS) + **JavaFX 17**.
  + PostgreSQL **15+** (con extensión pgcrypto para encriptación opcional).
* **Sistemas operativos:**
  + Windows 10/11 (64-bit).
  + Linux (distribuciones basadas en Debian/Ubuntu).

Mantenibilidad

* **Código Escalable:**
  + Estructura modular (separación clara entre lógica de negocio, acceso a datos e interfaz).
* **Base de datos:**
  + Scripts SQL versionados (en Git o similar).

Documentación

* **Técnica:**
  + Diagrama ER de la base de datos (usando draw.io).
  + Diccionario de datos (descripción de tablas/columnas clave).

## Mostrar reportes

Ficha Centro

* Nombre del centro
* Dirección postal del centro
* Logotipo del centro
* Teléfono y email de contacto
* Nombre del director general
* Nombre del jefe del departamento de recursos humanos
* Nombre del jefe del departamento de contabilidad
* Nombre del secretario general del sindicato

Ficha Conductor

* Nombre del conductor
* Número de documento de identidad
* Dirección, teléfono, correo
* Foto
* Numero de licencia
* Categoría licencia (moto, automóvil, camión, autobús)
* Restricciones (uso de lentes, conducción diurna, etc.)
* Estado de la licencia
* Puntos en la licencia
* Infracciones registradas (tipo, fecha, puntos deducidos)
* Fecha Emisión Licencia
* Fecha Vencimiento Licencia

Ficha Entidad Asociada

* Nombre de la entidad
* Tipo (autoescuela o clínica)
* Dirección
* Teléfono y email de contacto
* Nombre del director

Reporte de Licencias Emitidas en un Período de Tiempo

Se requiere **FECHA INICIO – FECHA FIN**

**Presentar la información ordenada por fecha de emisión.**

* Código de la licencia
* Nombre del conductor
* Tipo de licencia
* Fecha de emisión
* Fecha de vencimiento
* Estado de la licencia

Reporte de Exámenes Realizados en un Período de Tiempo

Se requiere **FECHA INICIO – FECHA FIN**

**Presentar la información ordenada por fecha de examen.**

* Código del examen
* Nombre del conductor
* Tipo de examen
* Fecha del examen
* Resultado del examen
* Entidad donde se realizó el examen

Reporte de Infracciones Registradas en un Período de Tiempo

Se requiere **FECHA INICIO – FECHA FIN**

**Presentar la información ordenada por fecha de infracción.**

* Código de la infracción
* Nombre del conductor
* Numero licencia
* Tipo de infracción
* Fecha de la infracción
* Lugar de la infracción
* Puntos deducidos
* Estado de la infracción (pagada o pendiente)

Reporte Consolidado de Infracciones por Tipo en un Año Determinado

Se requiere **Año**

**Presentar la información ordenada por tipo de infracción.**

* Tipo de infracción (leve, grave, muy grave)
* Cantidad de infracciones
* Puntos totales deducidos
* Multas totales pagadas
* Multas totales pendientes

Reporte de Conductores con Licencias Vencidas en un Período de Tiempo

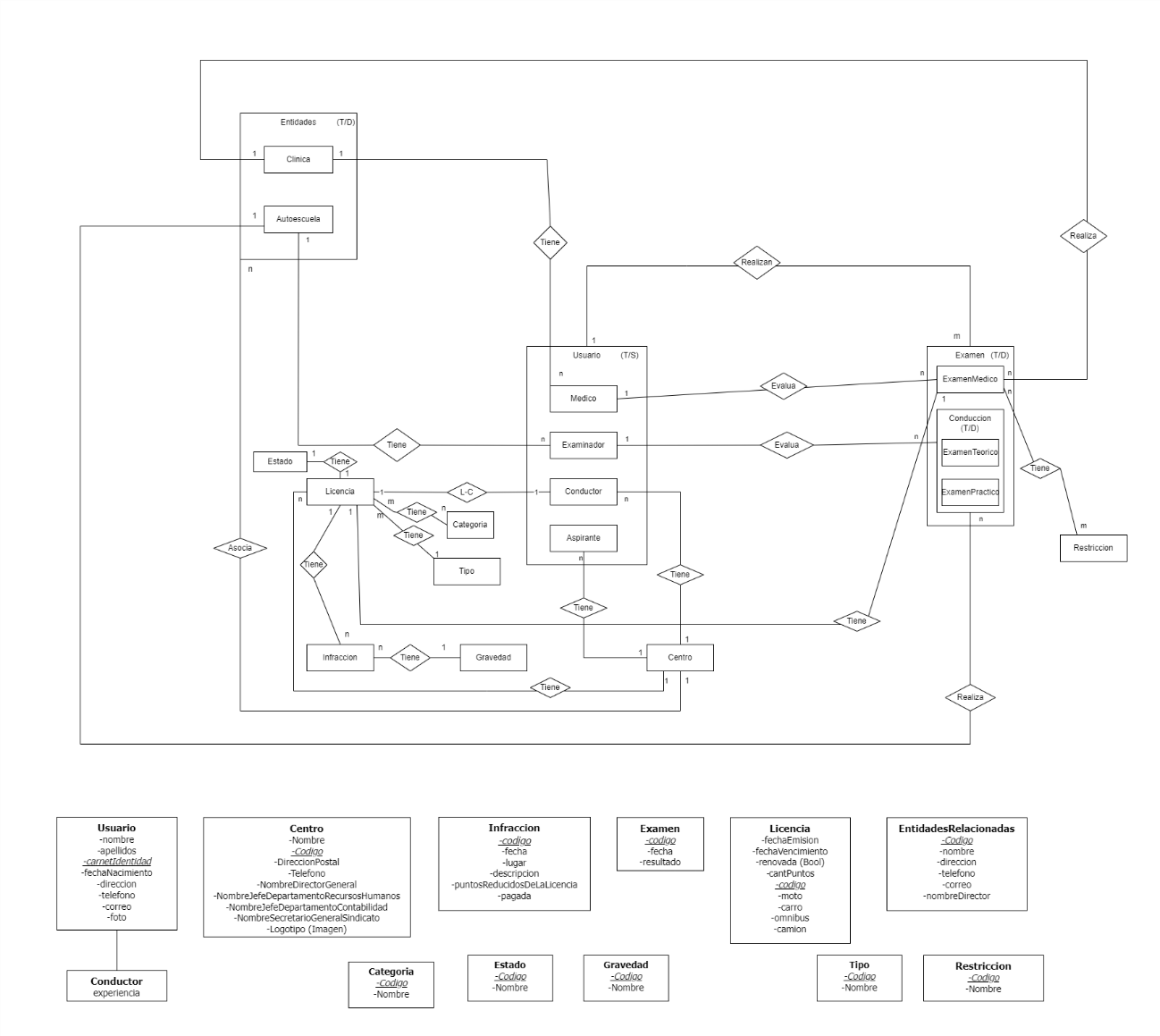
Se requiere **Tiempo que lleva vencida la licencia**

**Presentar la información ordenada por fecha de vencimiento.**

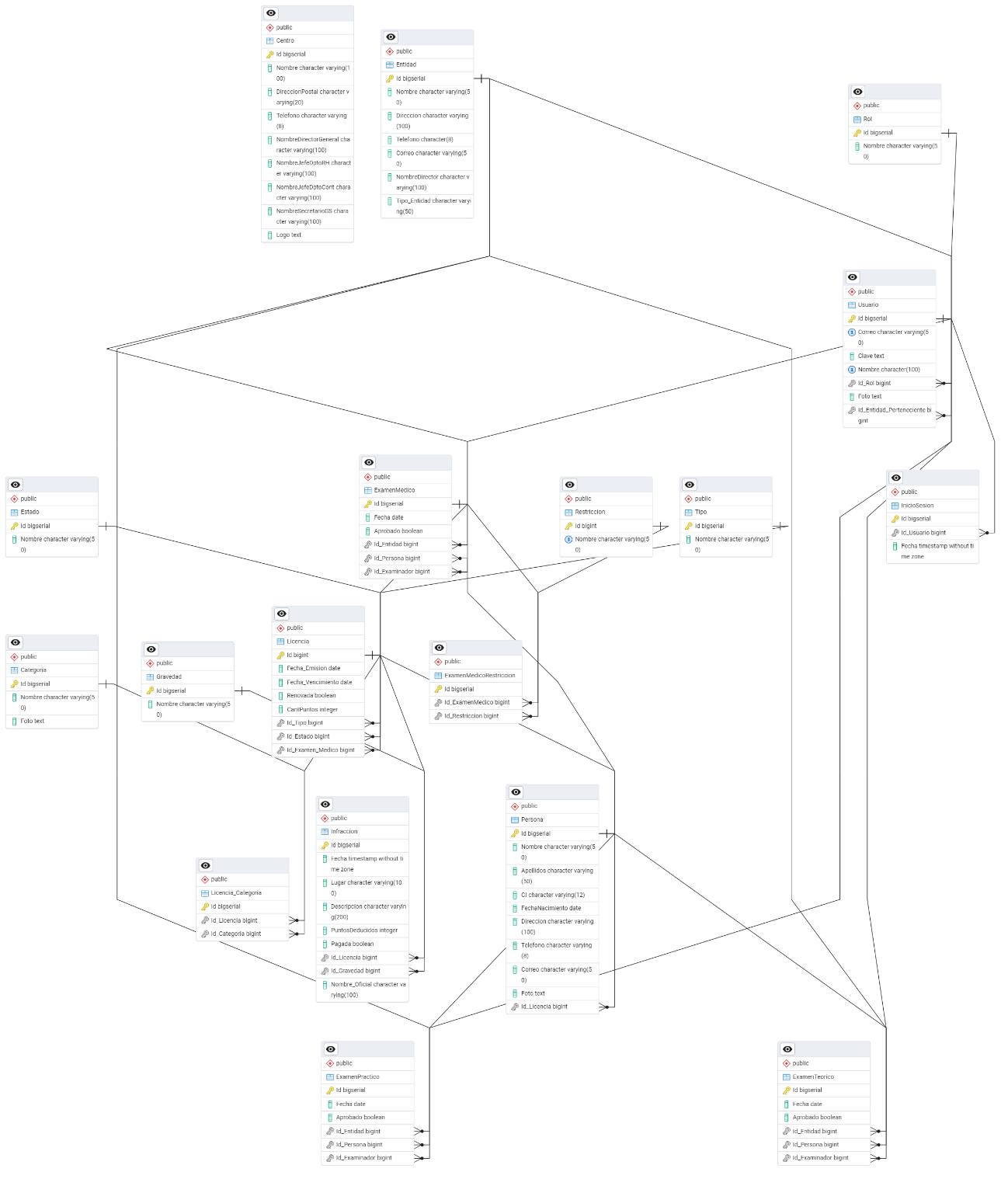
* Nombre del conductor
* Número de documento de identidad
* Número licencia
* Tipo de licencia
* Fecha de vencimiento
* Estado de la licencia

# Solución propuesta

## Diagrama entidad relación



## Diagrama físico



La solución implementada sigue un enfoque escalable y robusto que garantiza el correcto funcionamiento del sistema mediante la aplicación de **estándares de codificación y buenas prácticas de desarrollo**. El código está estructurado siguiendo principios de **código limpio (Clean Code)**, con métodos bien definidos, nombres descriptivos y responsabilidades únicas, lo que facilita su mantenimiento y extensión. Además, se han aplicado **formas normales (1FN, 2FN y 3FN)** en el diseño de la base de datos para eliminar redundancias y asegurar la integridad de los datos, optimizando así las consultas y evitando anomalías en las operaciones CRUD. La arquitectura en capas (presentación, lógica de negocio y datos) promueve la separación de preocupaciones, mientras que el uso de tecnologías estándar como **Java, PostgreSQL y JavaFX** asegura compatibilidad y estabilidad. Estas decisiones técnicas, combinadas con prácticas como la validación centralizada de datos y el manejo adecuado de excepciones, proporcionan una base sólida para escalar el sistema con nuevos módulos o funcionalidades sin comprometer su rendimiento o seguridad.

## Entidades principales

|  |  |
| --- | --- |
| Entidad | Atributos |
| Persona | Pk(Id) nombre, apellidos, ci, fechaNacimiento, dirección, teléfono, correo, foto, id\_licencia |
| Licencia | Pk(Id) fechaEmision, fechaVencimiento, renovada, cantPuntos, idTipo, idEstado, idExamenMedico |
| Infracción | Pk(Id) fecha, lugar, descripción, puntosDeducidos,pagada,idLicencia,idGravedad, nombreOficial |
| Entidad | Pk(Id) nombre,dirección,teléfono,correo,nombreDirector,tipoEntidad |
| ExamenMedico | Pk(Id) fecha, aprobado, identidad, idPersona, idExaminador, restricciones |
| ExamenTeorico | Pk(Id) fecha, aprobado, identidad, idPersona, idExaminador |
| ExamenPractico | Pk(Id) fecha, aprobado, identidad, idPersona, idExaminador, idExamenMedico |

## Relaciones

|  |  |  |  |
| --- | --- | --- | --- |
| Entidad | Relación | Entidad | Explicacion |
| Persona | 1:N | ExamenMedico | Una persona puede realizar muchos exámenes médicos |
| Persona | 1:N | ExamenTeorico | Una persona puede realizar muchos exámenes teóricos |
| Persona | 1:N | ExamenPractico | Una persona puede realizar muchos exámenes prácticos |
| Persona | 1:1 | Licencia | Una persona tiene una licencia y solo una |
| Licencia | M:N | Categoria | Muchas licencias tienen muchas categorías(Moto,Automovil ,ect) |
| Licencia | M:1 | Tipo | Muchas licencias tienen un tipo |
| Entidad | 1:M | Examen | Una entidad realiza muchos exámenes(de su especialidad) |
| Licencia | 1:M | Infracción | Una licencia tiene muchas infracciones |
| Infracción | M:1 | Gravedad | Muchas infracciones tiene una gravedad |
| ExamenMedico | M:N | Restricción | Muchos exámenes médicos tienen muchas restricciones |
| ExamenPractico | 1:1 | ExamenMedico | Un examen practico se relaciona con el examen medico correspondiente para obtener las restricciones |

# Arquitectura

Arquitectura N capas,

Back-End

Interfaz Usuario

Infraestructura

La arquitectura de tres capas (UI, Backend, Infraestructura) es ideal para este sistema offline de escritorio porque mejora la mantenibilidad, escalabilidad y organización del código. Al separar la interfaz de usuario (JavaFX), la lógica de negocio (Java) y la gestión de datos (PostgreSQL), se logra un código más limpio y modular, facilitando futuras actualizaciones o correcciones. Por ejemplo, si se cambia la base de datos o la interfaz, el núcleo de la aplicación (Backend) no requiere modificaciones. Además, esta estructura promueve la reutilización de componentes (como validaciones o consultas SQL) y aísla errores, ya que un fallo en la UI no afecta la lógica ni los datos. Aunque sea una aplicación offline, esta arquitectura asegura robustez y claridad, clave para un proyecto educativo donde el enfoque es aprender buenas prácticas de desarrollo.

# Seguridad

## Roles

|  |  |  |
| --- | --- | --- |
| **ROL** | **ACCESO AL SISTEMA** | **IMPLEMENTADO** |
| Administrador | SI | SI |
| Administrador-Autoescuela | SI | NO |
| Administrador-Medico | SI | NO |
| Trabajador-Autoescuela | SI | SI |
| Trabajador-Centro | SI | NO |
| Medico | SI | SI |
| Conductor | NO | --- |
| Aprendiz | NO | --- |

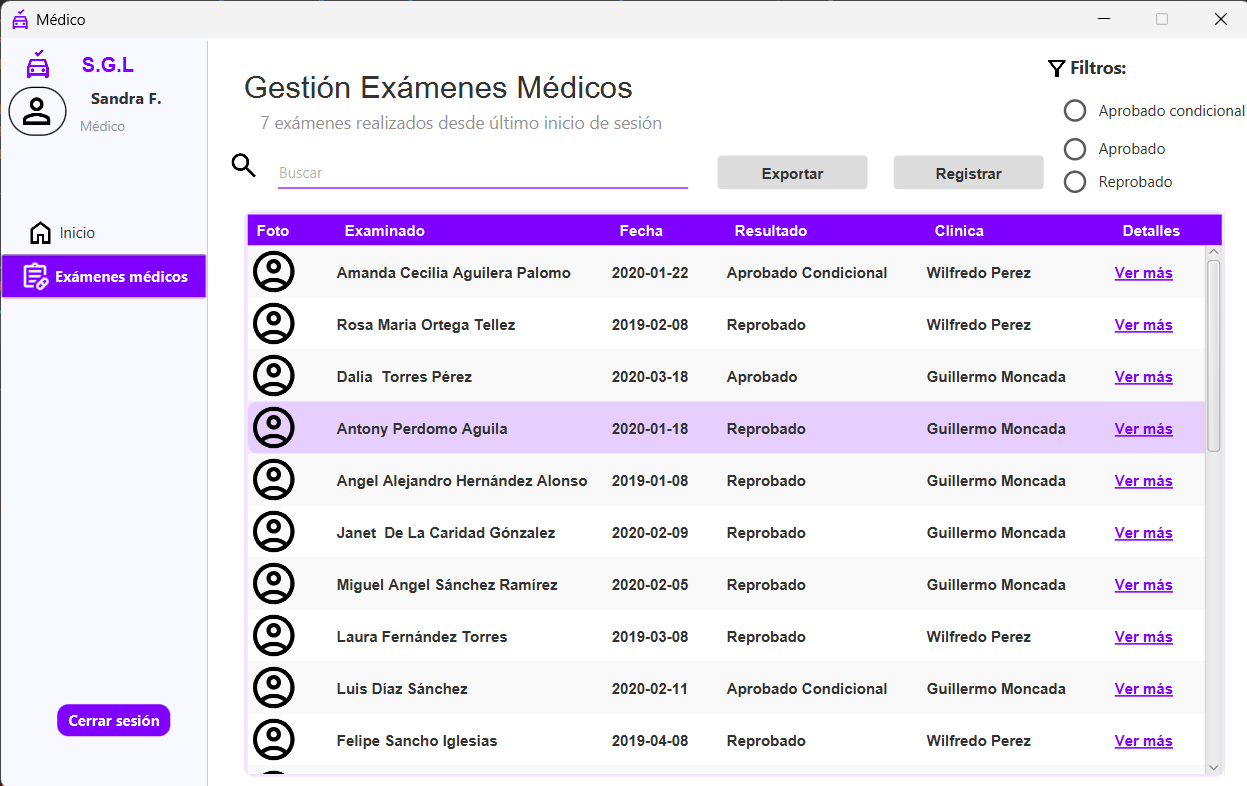
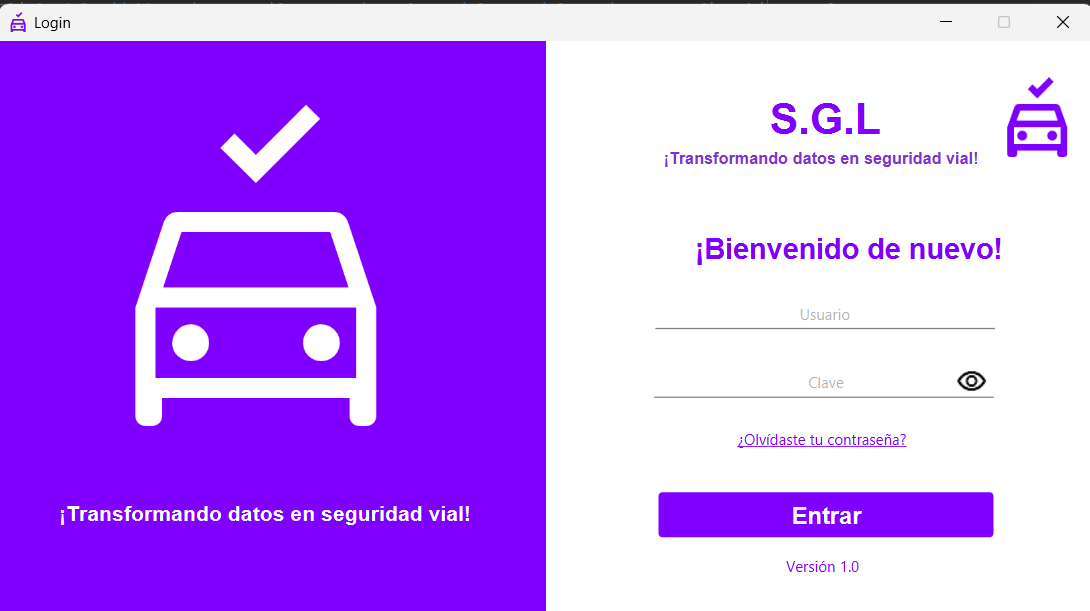
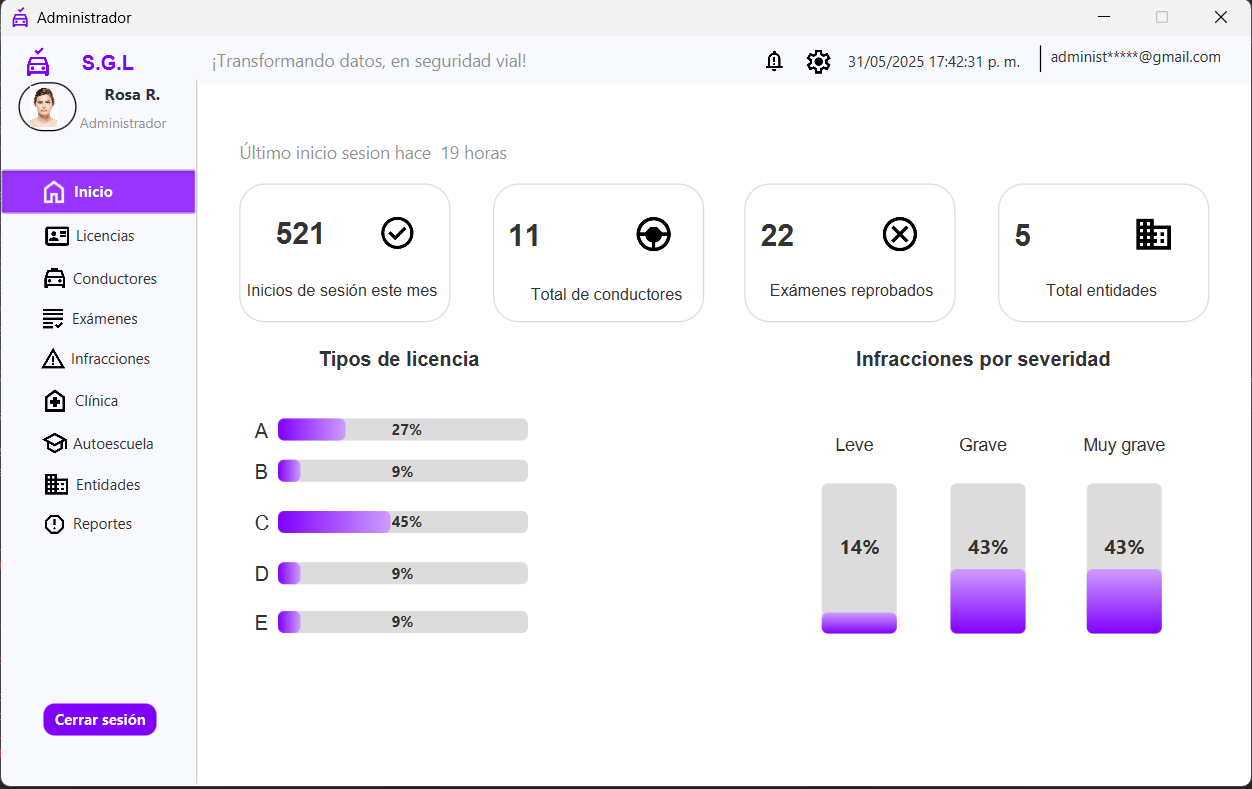
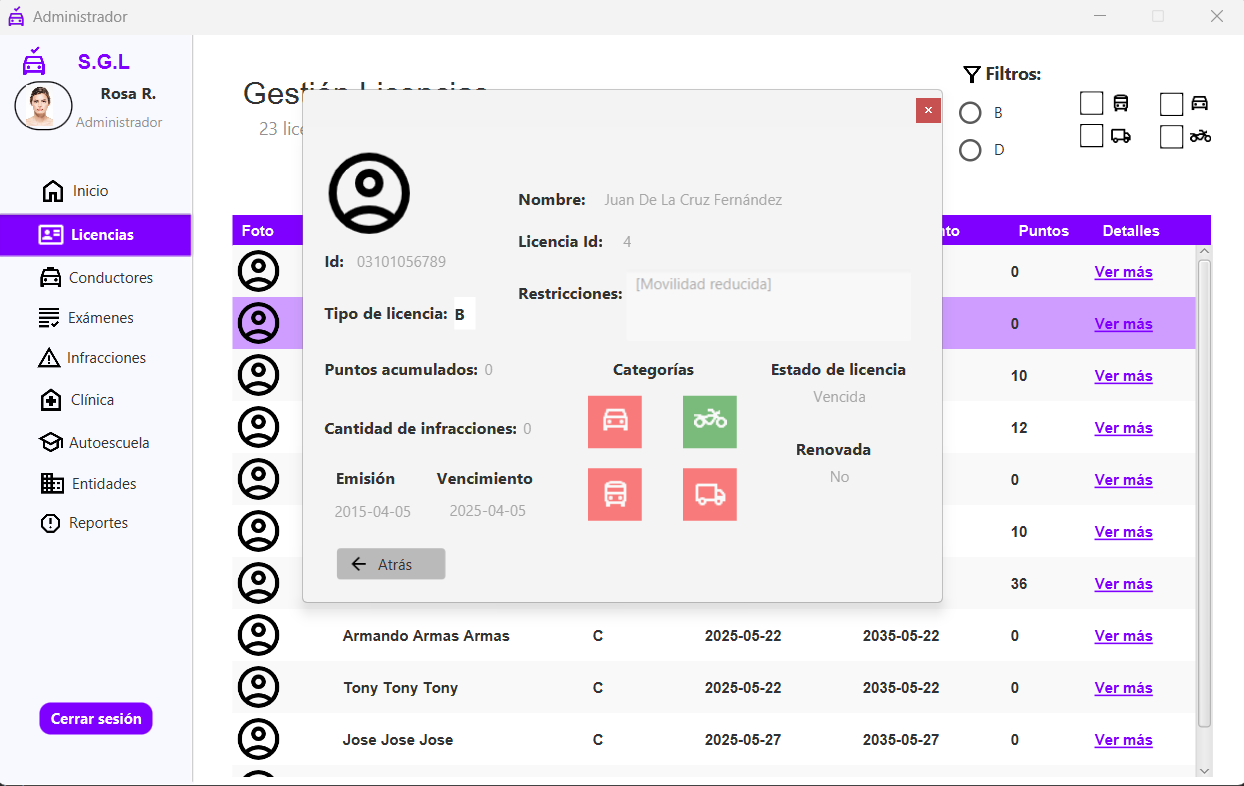
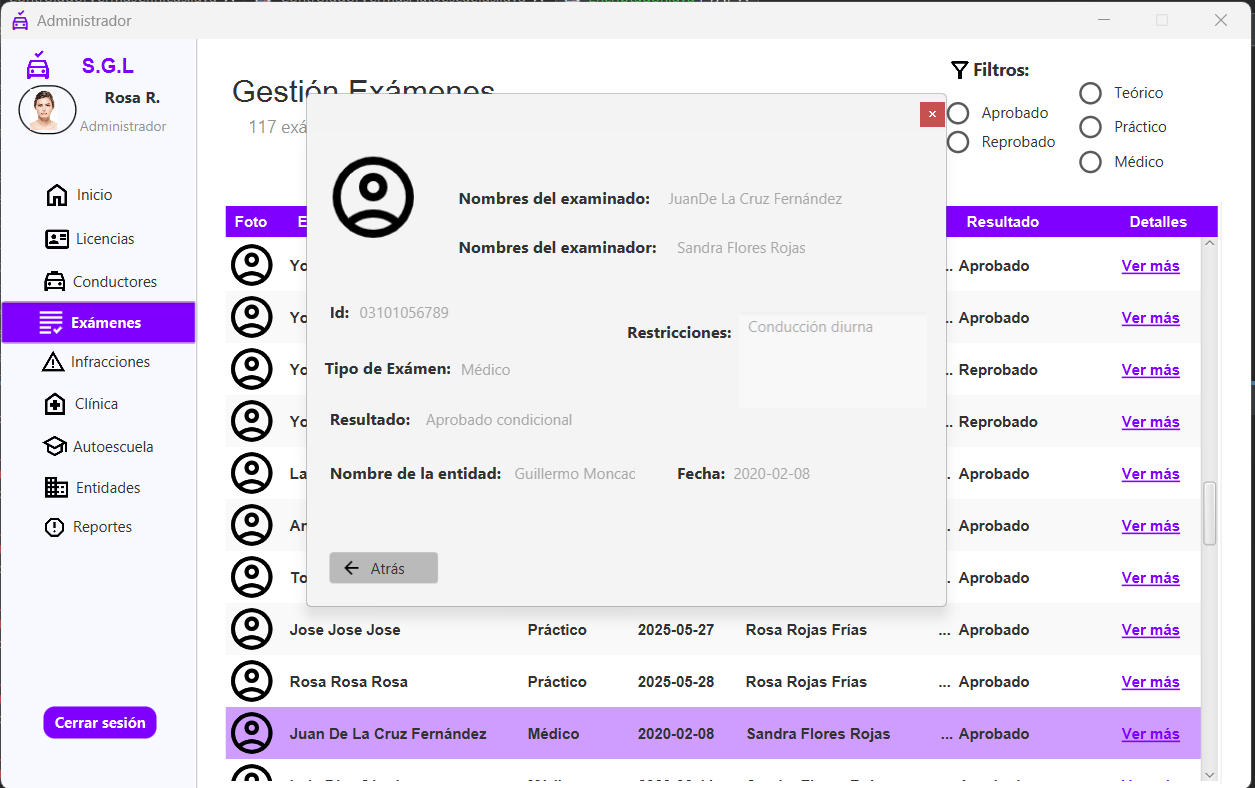
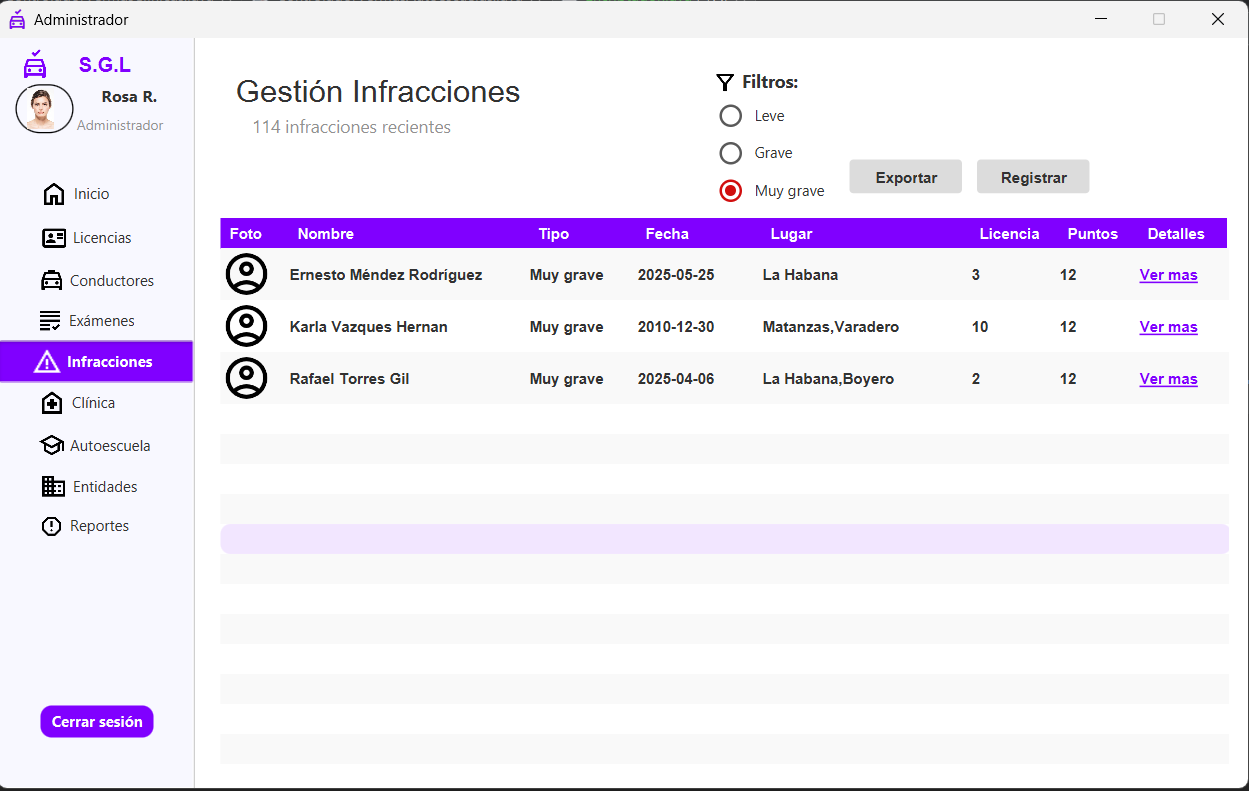
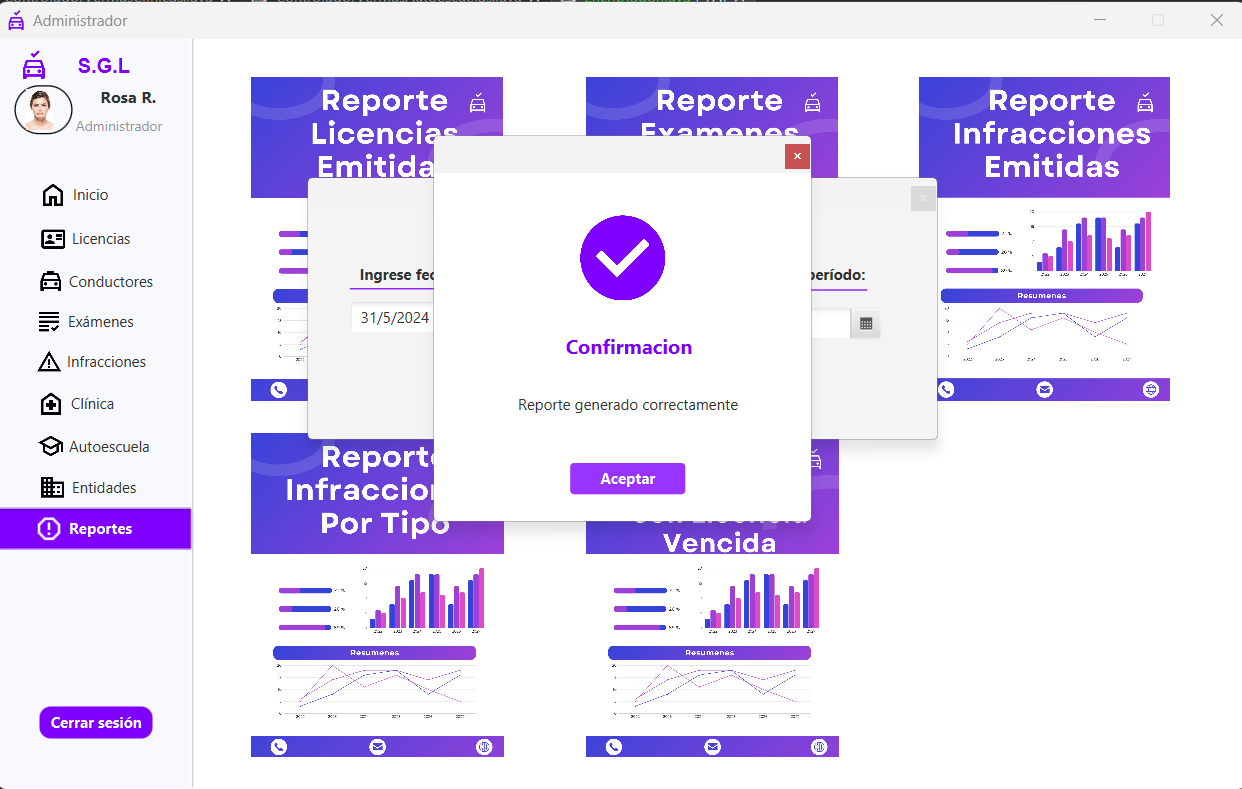
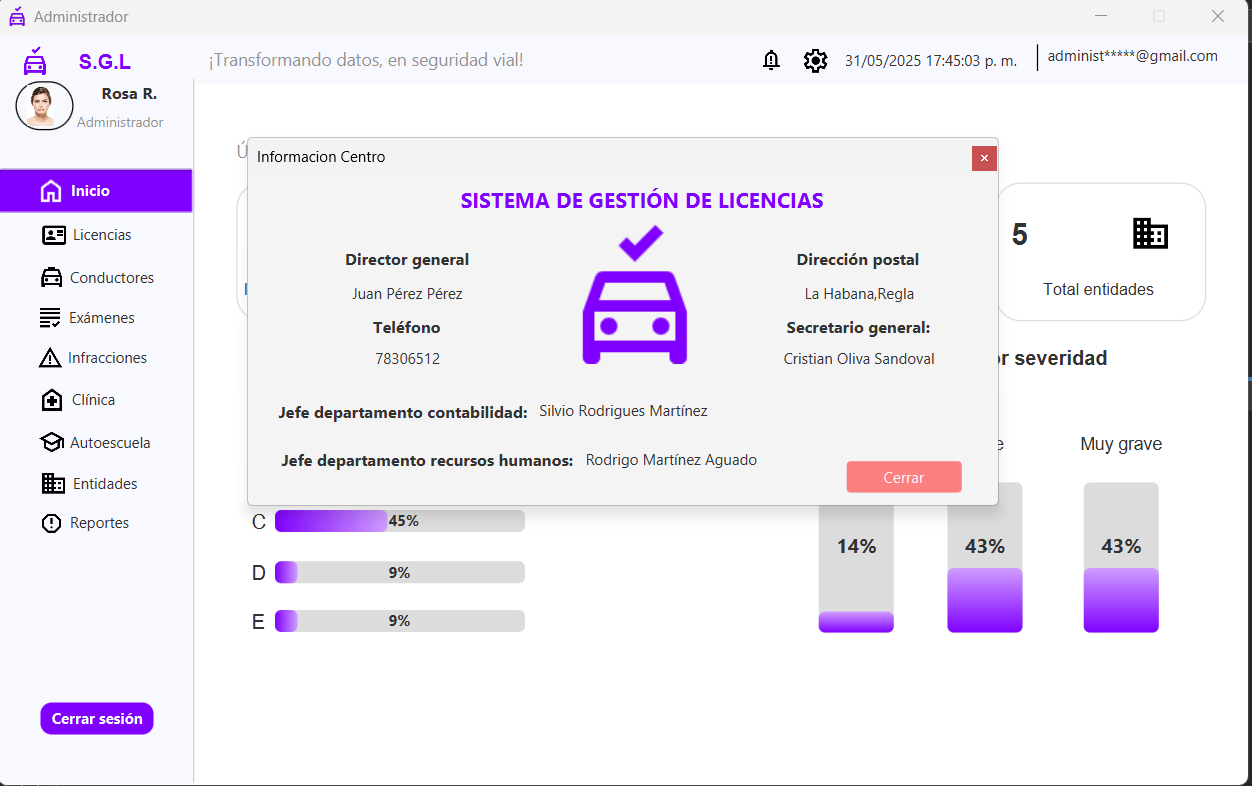
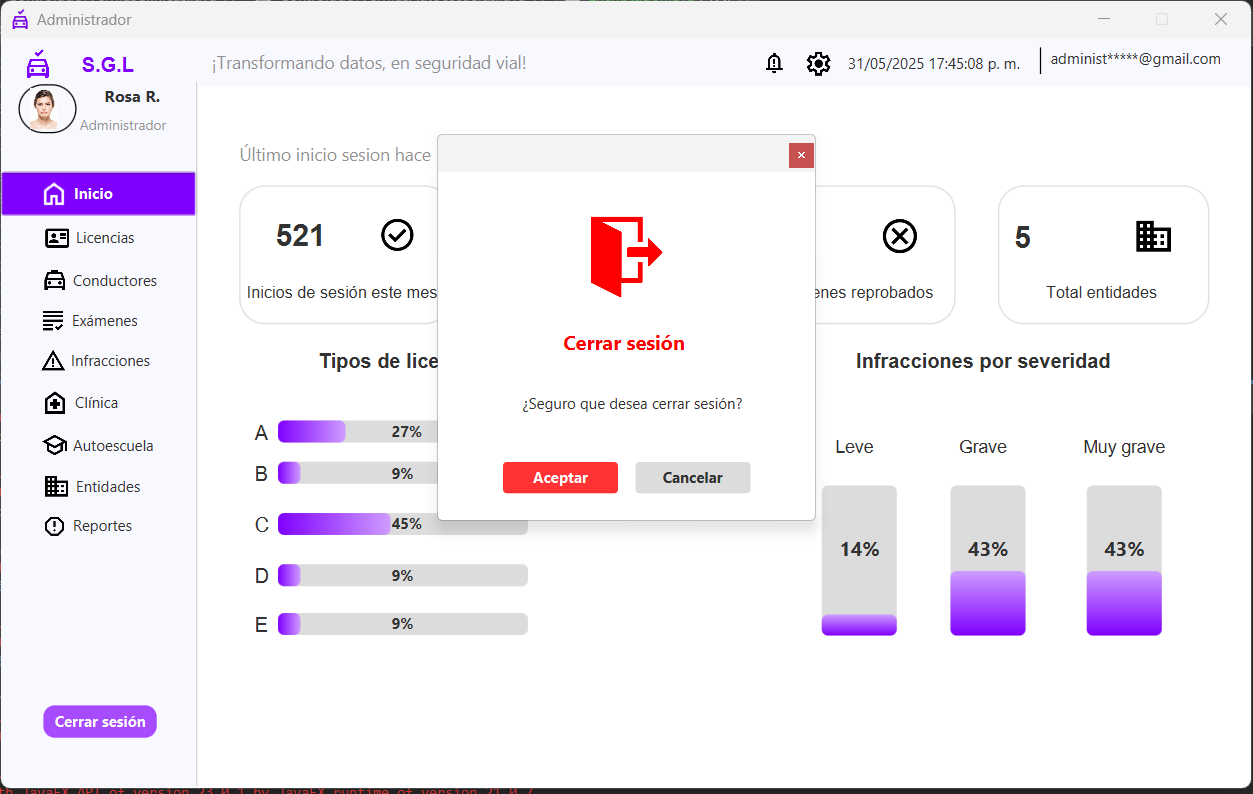
## Encriptación

Este método de cifrado por desplazamiento ampliado resulta ideal para fines educativos porque, aunque no ofrece seguridad real, proporciona una excelente introducción práctica a los principios fundamentales de la criptografía. Su simplicidad permite a los estudiantes comprender claramente conceptos como claves de encriptación, sustitución de caracteres y aritmética modular, sin verse abrumados por la complejidad matemática de algoritmos avanzados. Al trabajar con una lista definida de caracteres y un sistema de desplazamiento básico, los aprendices pueden visualizar fácilmente el proceso de transformación de datos, mientras que la estructura del código -con sus métodos separados para encriptar y desencriptar- ejemplifica buenas prácticas de programación como la modularidad y el manejo de casos límite. Particularmente valioso resulta para demostrar por qué los sistemas profesionales requieren técnicas más sofisticadas: al evidenciar sus vulnerabilidades (como la predictibilidad ante análisis de frecuencia), se comprende mejor la necesidad de mecanismos como el salting, las funciones de derivación de claves o el cifrado por bloques que emplean estándares como AES. En esencia, funciona como un "andador criptográfico" que, aunque insuficiente para aplicaciones reales, proporciona la base conceptual necesaria antes de abordar sistemas más robustos.

Original: admin123!

Encriptado: fsiqn678&

# Interfaz Usuario



# Tratamiento de errores

El sistema implementa una estrategia de propagación de errores donde las excepciones se generan en las capas inferiores (como acceso a datos o lógica de negocio) y ascienden progresivamente hasta la capa de presentación. Esta técnica centralizada permite que cada nivel maneje errores específicos de su dominio: la capa de infraestructura lanza excepciones técnicas (ej: SQLException), el backend las encapsula en excepciones de negocio con contexto significativo (ej: BusinessException), y finalmente la interfaz de usuario (JavaFX) las captura para mostrar mensajes claros y accionables al usuario mediante alertas o notificaciones visuales. Este enfoque mejora la trazabilidad de fallos (pues conserva el stack trace original), mantiene la separación de responsabilidades entre capas, y garantiza una experiencia de usuario consistente al estandarizar el formato de errores. Además, simplifica el debugging al preservar la cadena de causas desde el origen técnico hasta su representación final.

**El patrón ValidatorComposite** actúa como un contenedor inteligente que agrupa múltiples validadores individuales (cada uno implementando la interfaz IValidador) para ejecutarlos secuencialmente como una sola unidad de validación. Cada validador concreto (como NotEmptyValidator para campos vacíos o NumericValidator para formatos numéricos) implementa la interfaz IValidador, que define un método común (ej: validar(String input)). El ValidatorComposite recibe una lista de estos validadores y los ejecuta en orden: si alguno falla, interrumpe el proceso y retorna el error correspondiente, mientras que si todos pasan, la validación global es exitosa. Este enfoque es ideal para campos con reglas complejas (ej: un "CI" que debe ser no vacío, numérico y de longitud específica), ya que encapsula la lógica dispersa en componentes reutilizables y mantiene el código limpio. Al usar una interfaz común (IValidador), el sistema puede extenderse fácilmente añadiendo nuevos validadores sin modificar la lógica existente, siguiendo el principio **Open/Closed** de **SOLID**. Por ejemplo, en el sistema de licencias, este patrón garantiza que las validaciones de datos (como números de licencia o fechas) sean consistentes y mantenibles.