PROYECTO GESTOR ACADÉMICO

UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

PRESENTADO POR:
OSCAR MANUEL CONTRERAS GACHA – 20221020052
NICOLÁS ROMERO RODRÍGUEZ – 20222020023

PRESENTADO A:
Henry Alberto Diosa

UNIVERSIDAD DISTRITAL FRANCISCO JOSÉ DE CALDAS
FACULTAD DE INGENIERÍA
FUNDAMENTOS DE INGENIERÍA DE SOFTWARE
2025

Versión	Fecha de Revisión	Autor(es)	Descripción del Cambio
0.1	2025-09-29	-Oscar Manuel Contreras Gacha -Nicolás Romero Rodríguez	Desarrollo Funcional
0.2	2025-11	-Oscar Manuel Contreras Gacha -Nicolás Romero Rodríguez	Correcciones del desarrollo Funcional

PROYECTO SEMESTRAL 2025-III

1. INTRODUCCIÓN

El presente documento expone el diseño, análisis y especificación del *Gestor Académico*, un sistema de software orientado a apoyar los procesos administrativos y formativos de un pequeño colegio que maneja los grados de Párvulos, Caminadores y Prejardín. Con el fin de responder a las necesidades de la institución, se desarrolló un modelo integral que abarca desde la identificación del problema, la formalización de los requerimientos funcionales y no funcionales, la definición de interfaces, y la construcción de modelos estructurales y dinámicos que definen el comportamiento del sistema en su totalidad.

El proyecto se fundamenta en los principios de la Ingeniería de Software, siguiendo una arquitectura basada en el patrón Modelo–Vista–Controlador (MVC) y en el uso de Java con JPA para la gestión de persistencia. A través de este documento se presentan los diferentes artefactos generados durante el proceso de análisis y diseño, tales como el modelo de dominio, los casos de uso, las tablas de requerimientos, los bocetos de interfaz y el diagrama relacional. Cada uno de estos elementos contribuye a garantizar que la solución propuesta sea consistente, escalable, trazable y alineada con las necesidades reales de los usuarios: acudientes, profesores, dirección y superusuario.

El propósito principal es proveer una herramienta que permita gestionar eficientemente cursos, estudiantes, logros académicos y preinscripciones, manteniendo al mismo tiempo un flujo de información claro, seguro y adaptable a la operación institucional. Este documento sirve como base para la futura implementación y despliegue del sistema.

CAPITULO 1 MODELO FUNCIONAL

2. DESCRIPCIÓN DEL PROBLEMA:

Un pequeño colegio con tres grados (Párvulos, Caminadores y Prejardín) requiere una aplicación de gestión académica que pueda ser utilizada por los acudientes, los profesores y la dirección de la institución para obtener las siguientes funcionalidades:

1. El inicio del sistema tendrá las opciones para llenar un **formulario de preinscripción** el cual puede ser usado por **cualquier usuario** y un **inicio de sesión** el cual desplegará las funciones de la aplicación dependiendo del r**ol de acceso**, los campos serán **nombre de usuario, contraseña y rol**, Un usuario puede tener **más de un rol.**
2. Para la **creación e inhabilitación** de usuarios solo podrá ser gestionado por el **superusuario** de la aplicación.
3. Al momento de iniciar sesión solo se **habilitarán** las funcionalidades designadas del rol específico que accedió.
4. El formulario **permite registrar** los datos personales básicos de los acudientes (Nombres, apellidos, datos de contacto) y un **perfil básico del niño(a)** (Nombres, apellidos, edad, grado al que aplica) que aspira a ser cuidado y formado en la institución.
5. Si el niño es aceptado, la dirección **creará** el usuario correspondiente para el acudiente **añadiendo** el perfil del estudiante a su usuario, o si **ya posee** un usuario solo **añadirá** el perfil del niño a la cuenta del usuario para la relación acudiente-estudiante.
6. Cuando se realiza una preinscripción se **guarda en la base de datos para su futura consulta.**
7. Al momento de ser aceptado el estudiante, será asignado al grado al cual aspiro al momento en que se creen los grupos.
8. El sistema solo manejará los grados Párvulos, Caminadores y Prejardín, pero para cada uno de estos se puede **crear cursos** con un mínimo de 2 cursos por grado.
9. Gestionar, por parte de la dirección del colegio, las **categorías de logros** para la evaluación de cada estudiante. Las categorías recomendadas son:
    a. Logros psicosociales.
    b. Logros psicomotores.
    c. Logros cognitivos.
    d. Logros procedimentales.
10. Para la creación de un curso, el **mínimo** de estudiantes para este curso es de 5 y con **máximo** de 10 estudiantes.
11. Cada curso tendrá **un profesor** especial para este, y este será el **profesor director** de este curso y será el único profesor para este curso.
12. Los profesores de cada grupo pueden consultar **los listados** de clase
13. Los Profesores para su curso asignado tiene la opción de la **gestión de logros** por cada categoría de acuerdo a su criterio profesional, además de **generar los boletines** por periodo.
14. Los profesores pueden **realizar consultas** de históricos de calificaciones para un estudiante con posibilidad de **generar un reporte**.

3. REQUERIMIENTOS DE INTERFACES

3.1 Interfaces de Usuario (UI)

El sistema debe proporcionar interfaces gráficas intuitivas, consistentes y accesibles para los distintos roles del sistema (Administración, Profesor, Superusuario y Usuarios en Preinscripción). Las interfaces deben estar desarrolladas en **Java Swing**, seguir el patrón **MVC**, y garantizar una interacción fluida entre el usuario y las funcionalidades del sistema.

Requerimientos principales de UI

1. **Login y selección de rol**
    a. El sistema debe permitir que el usuario ingrese usuario, contraseña y rol.
    b. Debe validar credenciales y redirigir automáticamente a la vista correspondiente al rol.
2. **Gestión de Grados (Rol Administración)**
    a. Mostrar el listado completo de grados.
    b. Permitir seleccionar un grado para visualizar los cursos asociados.
    c. Proveer botones para crear cursos y asignar estudiantes a un curso.
3. **Gestión de Cursos por Grado**
    a. Mostrar cursos disponibles en el grado seleccionado.
    b. Permitir crear nuevos cursos.
    c. Permitir asignar estudiantes a cursos existentes.
    d. Actualizar la interfaz automáticamente cuando haya cambios en la persistencia.
4. **Vista del Profesor (Rol Profesor)**
    a. Mostrar la información del curso asignado.
    b. Listar estudiantes inscritos al curso.
    c. Incluir un botón para gestionar logros del curso.
5. **Gestión de Logros por Categoría**
    a. Mostrar botones que permiten cambiar entre categorías de logro.
    b. Listar los logros de la categoría seleccionada.
    c. Ofrecer botones para crear y eliminar logros.
6. **Preinscripción (Acceso sin login)**
    a. Formularios para que estudiantes diligencien información.
    b. Validación y almacenamiento de datos en el sistema.

Requisitos generales de diseño

- El sistema debe mantener coherencia visual en fuentes, colores y distribución.

- Las vistas deben ser responsivas dentro del entorno Swing (uso adecuado de layouts).
- Los mensajes al usuario deben ser claros, evitando errores silenciosos.
- Las ventanas deben abrirse centradas y con tamaños adecuados a su contenido.
- Todo elemento seleccionable debe estar correctamente habilitado y deshabilitado según el contexto.

3.2 Interfaces de software (SW)

Las interfaces de software definen cómo interactúan los distintos componentes internos del sistema, así como la comunicación entre las capas de la arquitectura.

Requerimientos de interacción interna

1. **Controladora ↔ Vista**
    a. Las vistas deben invocar métodos de la `Controladora` para realizar operaciones del sistema.
    b. No debe existir lógica de negocio en las vistas.
2. **Controladora ↔ ControladoraPersistencia**
    a. La Controladora debe enviar objetos de dominio a la capa de persistencia.
    b. Debe gestionar la creación, actualización y consulta de entidades tales como:
    Grado, Curso, Profesor, Estudiante, Logro, CategoríaLogro, Token.
3. **Capa de Persistencia ↔ JPA (EntityManager)**
    a. La persistencia debe abstraer todas las operaciones CRUD.
    b. Debe utilizar clases JPA (DAO/JPAController) para manipular entidades.
    c. Los métodos deben asegurar el manejo del EntityManager y transacciones.
4. **Modelo ↔ Persistencia (Entidades)**
    a. Todas las entidades deben estar anotadas con JPA:
        i. `@Entity`, `@Id`, `@GeneratedValue`, `@OneToMany`, `@ManyToOne`, `@ManyToMany`, `@OneToOne`.
    b. Las relaciones deben reflejar correctamente el modelo conceptual:
        i. Un profesor tiene un curso.
        ii. Un curso tiene un grado, un profesor, múltiples estudiantes y múltiples categorías.
        iii. Una categoría tiene múltiples logros.
        iv. Un logro pertenece a una categoría.

Intercambio de datos

- Las listas deben devolverse siempre como `List<T>`, nunca `null`.
- Los métodos deben manejar excepciones controladamente.
- El sistema debe permitir consultas particulares como:
    - Obtener estudiantes sin curso.
    - Obtener profesores sin curso.
    - Obtener cursos por grado.
    - Obtener logros por categoría.
    - Validar credenciales por token.

3.5 Persistencia (PERS)

La capa de persistencia es responsable del almacenamiento permanente de datos y de garantizar la integridad de las entidades del dominio. El sistema utiliza **JPA (Java Persistence API)** con `EntityManager` para gestionar todas las operaciones de base de datos.

Requerimientos de persistencia

1. **Gestión de Entidades**
    a. Las entidades deben mapearse completamente mediante anotaciones JPA.
    b. Las relaciones deben respetar los tipos definidos:
        i. Uno-a-Muchos
        ii. Muchos-a-Uno
        iii. Muchos-a-Muchos
        iv. Uno-a-Uno
    c. Las colecciones deben inicializarse como `ArrayList<>` para evitar `null`.
2. **Operaciones CRUD**
   La capa de persistencia debe proporcionar métodos para:
    a. Crear entidades (create)
    b. Editar/actualizar entidades (edit)
    c. Eliminar entidades (destroy)
    d. Buscar por ID o por atributos específicos
3. **Consultas especializadas (Queries)**
   Deben existir métodos para:
    a. Obtener profesores sin curso
    b. Obtener estudiantes sin curso

c. Obtener cursos por grado
d. Validar inicio de sesión por Token
e. Obtener logros según categoría
f. Buscar estudiantes o profesores por nombre o ID

4. **Manejo del EntityManager**
    a. Cada operación debe abrir y cerrar adecuadamente el EntityManager.
    b. Las transacciones deben estar controladas explícitamente.
    c. Se deben capturar excepciones y registrar errores para depuración.
5. **Integridad de Datos**
    a. Las relaciones deben mantenerse sincronizadas entre entidades.
    b. Toda operación que modifique relaciones debe actualizar ambos lados cuando sea necesario.
        i. Ejemplo: asignación de curso a profesor.
    c. Las operaciones que dependan de la existencia de registros deben validar previamente.
6. **Persistencia y Vistas**
    a. Toda creación o eliminación debe reflejarse automáticamente en las vistas mediante recarga de listas.
    b. La persistencia debe prevenir estados inconsistentes que afecten a la UI.

4. REQUERIMIENTOS FUNCIONALES

A continuación, se presenta el sistema de codificación utilizado para los requerimientos funcionales del proyecto de gestión académica. Este sistema ha sido diseñado para ser **único, claro y universal**, permitiendo una fácil identificación, trazabilidad y gestión de cada requerimiento a lo largo de su ciclo de vida.

Cada código de requerimiento sigue el siguiente formato estructurado:
código: <módulo>-<identificador>-<prioridad>

Módulos

Gestión de Usuario: **GU**
Formulación de preinscripción: **FP**
Gestión de Grados: **GG**
Perfil de estudiante: **PE**
Herramientas Dirección: **HD**

Identificador

Se trata de un número incremental de tres dígitos (ej., 001, 002, 003, ...) que asigna un código inmutable a cada requerimiento. Este número es único en todo el proyecto y no cambia si el requerimiento se reordena o si otros son eliminados.

Prioridad

Utiliza un código de una o dos letras
CR (Crítico): El proyecto no puede funcionar sin él.
AL (Alto): Es fundamental, pero el proyecto puede lanzarse con una funcionalidad parcial.
ME (Medio): Deseable, pero puede posponerse para una fase posterior.
BA (Bajo): Es un "plus" o mejora, no esencial para el funcionamiento.

Ejemplo de lectura del código

Un requerimiento con el código `GU-005-AL` se lee de la siguiente manera:

- `GU`: Pertenece al módulo de **G**estión de **U**suario.
- `005`: Es el quinto requerimiento funcional del proyecto.
- `AL`: Tiene una prioridad **A**lta, siendo fundamental para la funcionalidad del sistema.

4.1 TABLAS DE REQUERIMIENTOS

En esta sección se presenta un listado detallado de los requerimientos del sistema, organizados por módulos funcionales. Cada requerimiento incluye un código de identificación, un título descriptivo, el tipo de usuario al que está dirigido y una explicación de la funcionalidad.

Iniciar de aplicación		
Código	**Título**	**Descripción**
IA-009-AL	Diligenciar Preinscripción	Permite a cualquier usuario llenar y enviar el formulario de preinscripción,
IA-032-AL	Iniciar Sesión	El Usuario tendrá la opción iniciar sesión al iniciar la aplicación

Gestión de Usuarios

Código	Título	Descripción
GU-001-AL	Crear usuarios	Crear perfiles de usuariocon datos usuario y contraseña para ingreso al sistema
GU-002-AL	Consultar usuario	Mostrar información de los usuarios en modo lectura nombre, id, rol.
GU-004-AL	inhabilitar usuario	Dar la opción de inhabilitar a usuarios según sea necesario
GU-005-AL	Asignar roles	Identificar los roles para el usuario y asignarlos
GU-006-AL	Validación de usuario	Debe identificarse el acceso del usuario

Gestión de Grados		
Código	**Título**	**Descripción**
GG-012-AL	Crear cursos	Tener la posibilidad de agregar más cursos a los grados. Para la creación de un curso, el mínimo de estudiantes es de 10 y con máximo de 20 estudiantes
GG-013-AL	Asignar estudiantes	**Si el estudiante es aceptado** poder integrar al estudiante al grado que aspiro y un curso específico del grado
GG-014-AL	Generar Listados	Generar los listados de estudiantes por cursos
GG-016-AL	Crear categorías de logros	Permitir a la gestión de categorías de logros evaluativos para todos los grados
GG-017-AL	Crear logros	Habilitar la opción de la Gestionar de logros por cada categoría de acuerdo con su criterio profesional
GG-018-ME	Borrar logros	El profesor tiene la opción de borrar un logro si esta fuera de la franja evaluativa

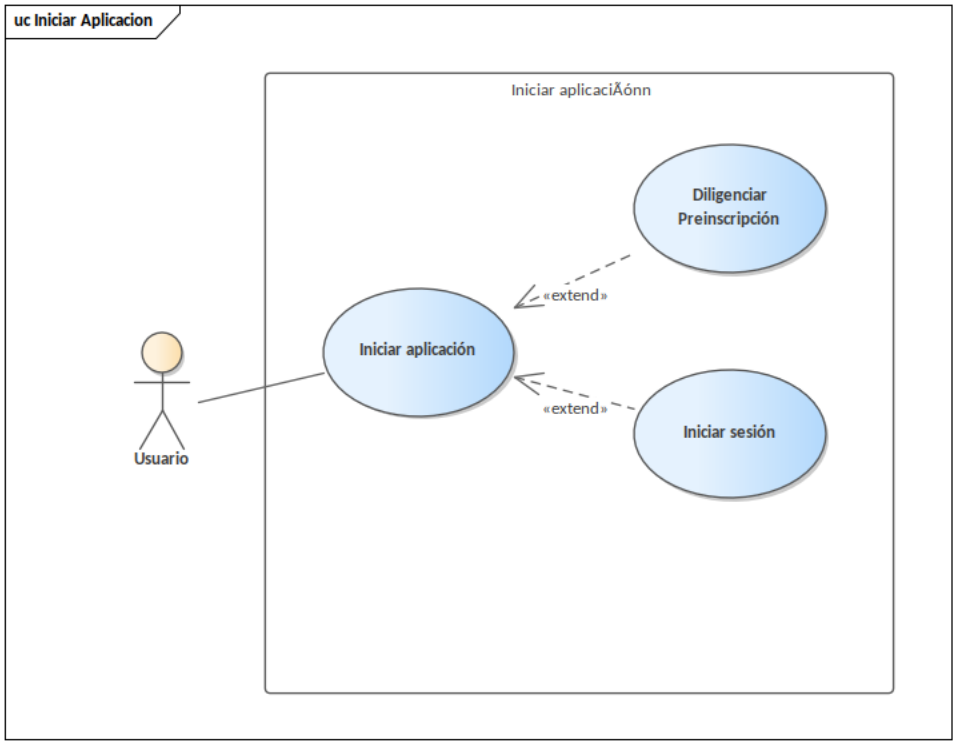
Gestión de Historial Académico		
Código	**Título**	**Descripción**
PE-020-AL	Registrar logros del estudiante	Habilitar a los profesores la evaluación de logros por estudiante Por período con tiempo de 2 meses para su registro

PE-021-AL	Consultar histórico de calificaciones	Consultar el historial de calificaciones del estudiante
PE-022-AL	Generar reporte Histórico	Posibilidad de generar el reporte del historial de calificaciones
PE-023-AL	Consultar logros	El acudiente puede consultar los logros de el/los estudiantes que representa
PE-024-ME	Generar reporte de logros	El profesor podrá generar un reporte de logros consultados

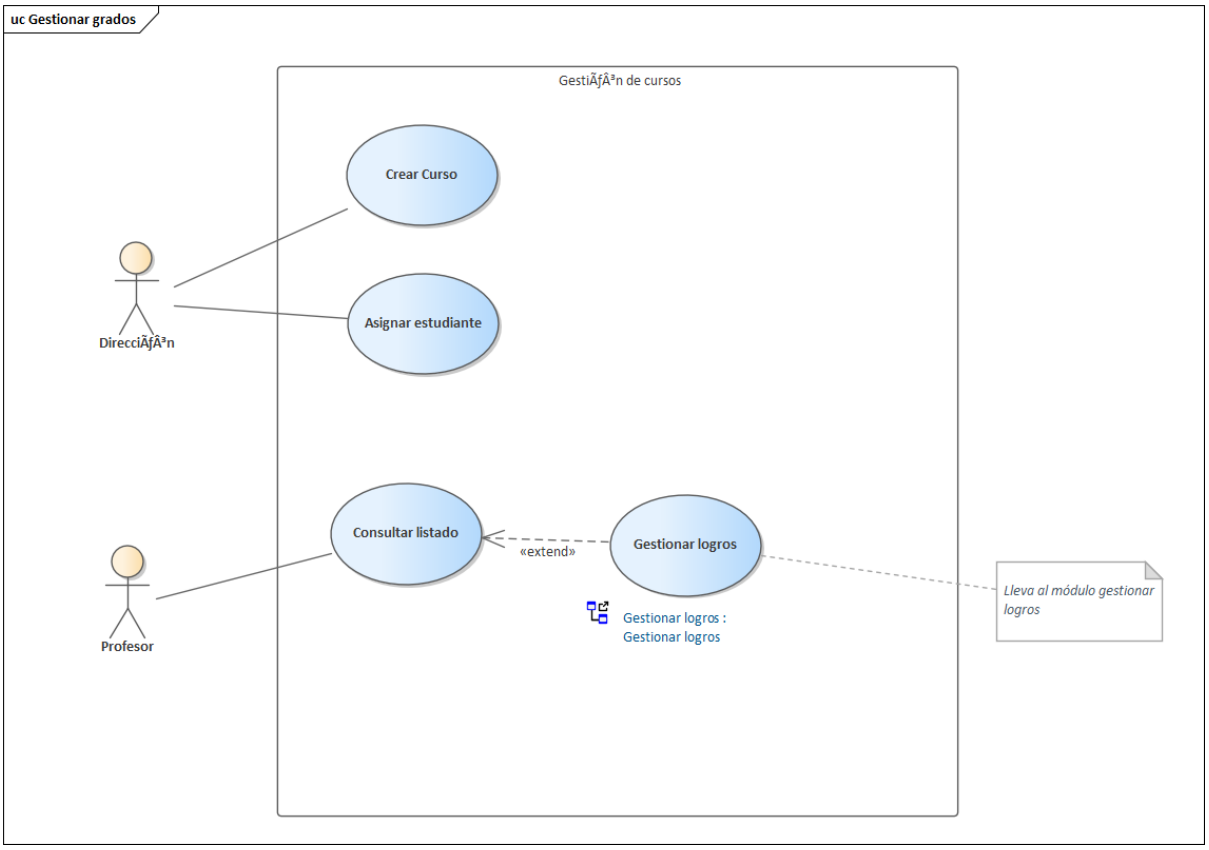
5. CASOS DE USO

5.1 Diagrama de casos de uso

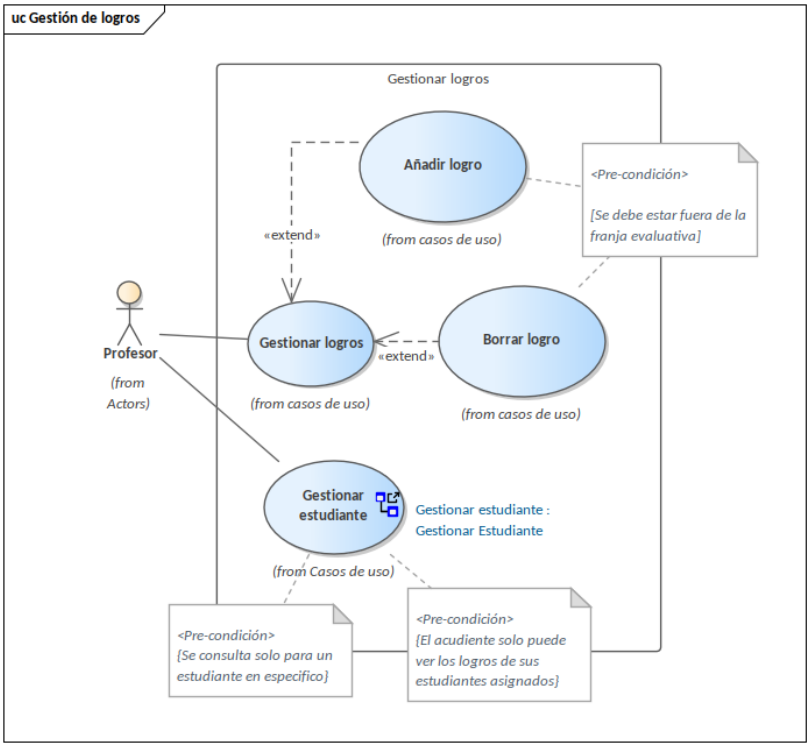
5.1.1 Inicio de Aplicación



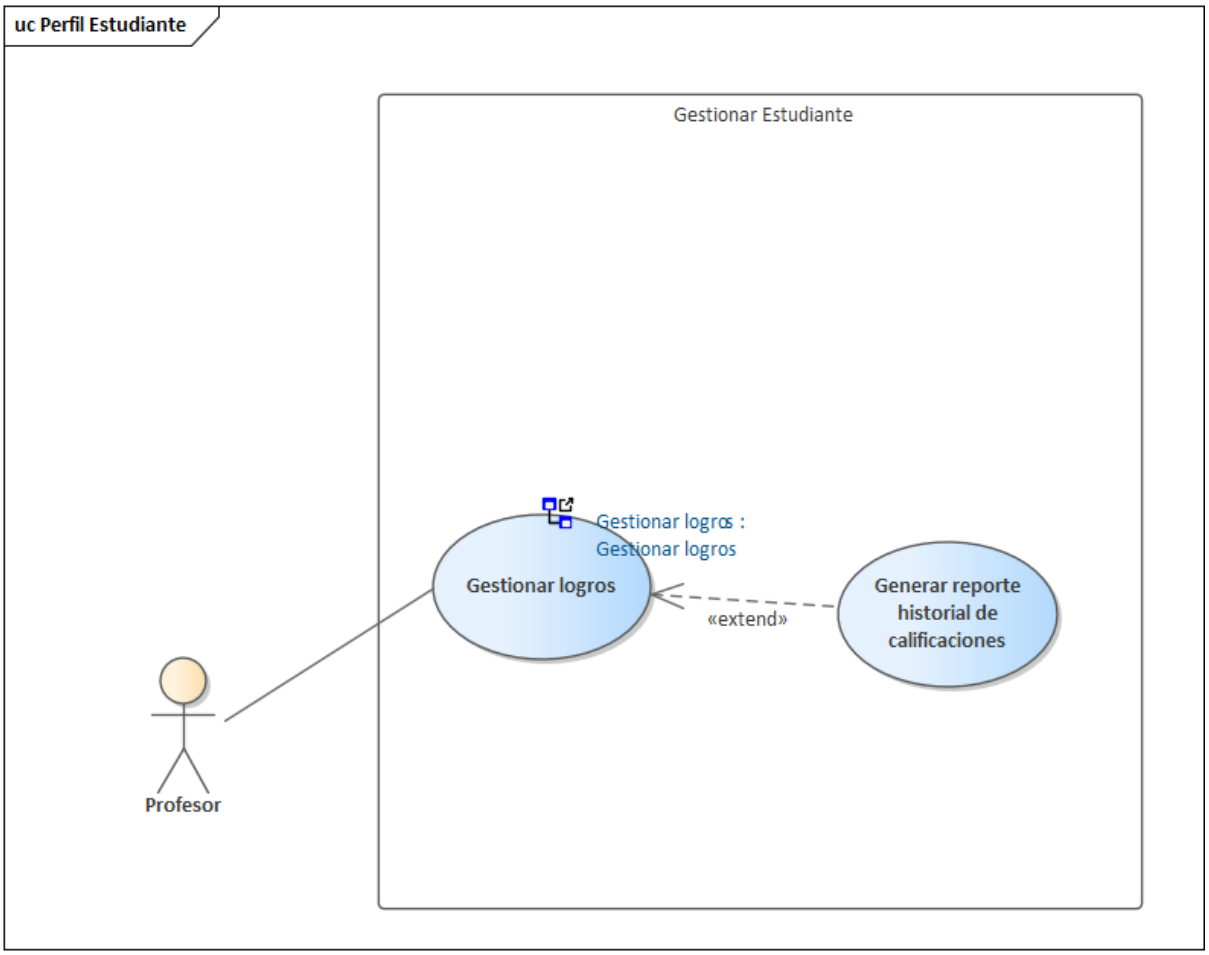
5.1.2 Gestión de cursos



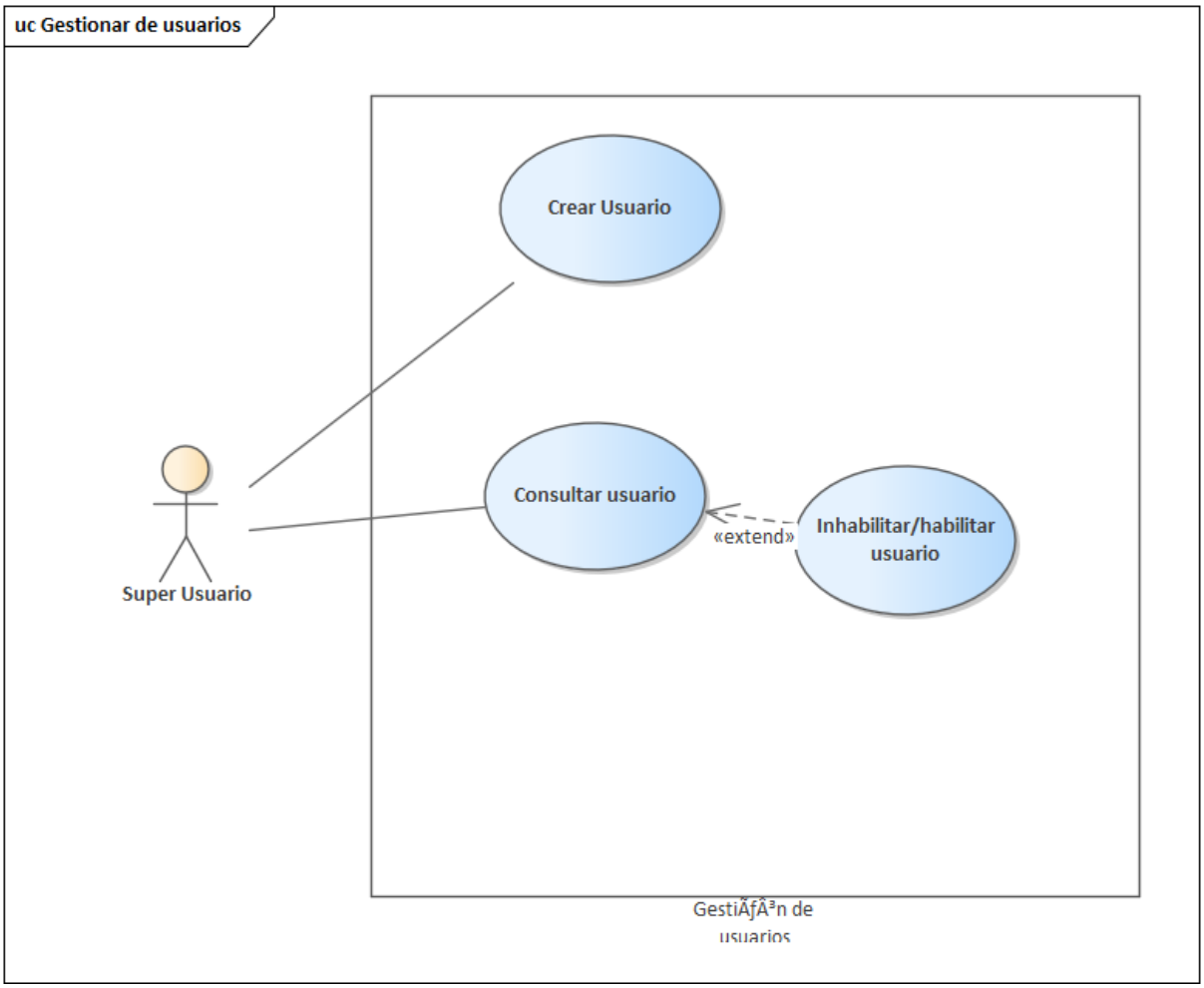
5.1.3 Gestión de logros



5.1.4 Gestionar Estudiante



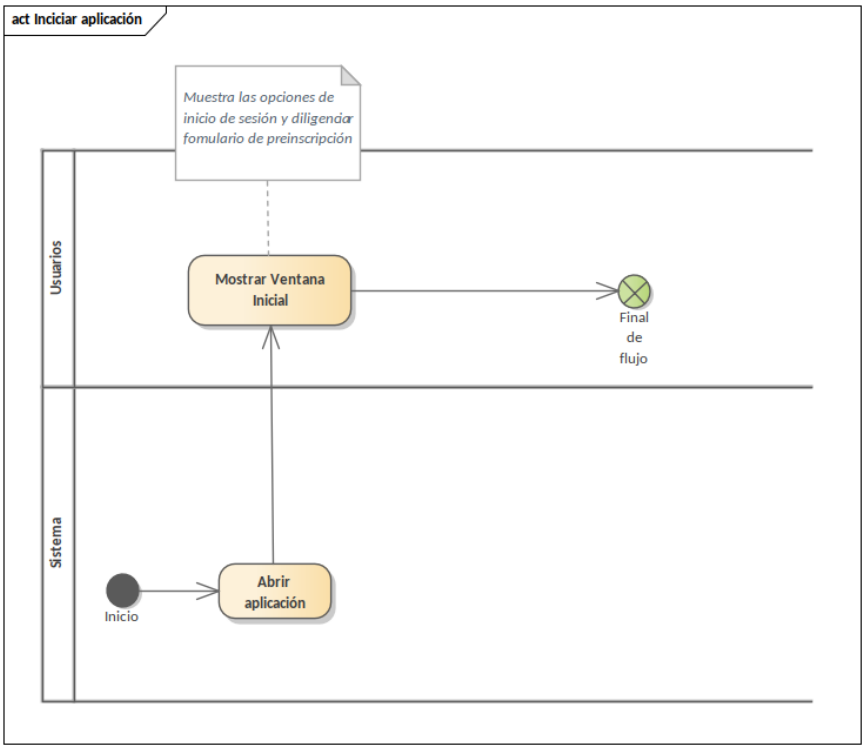
5.1.5 Gestión de Usuarios



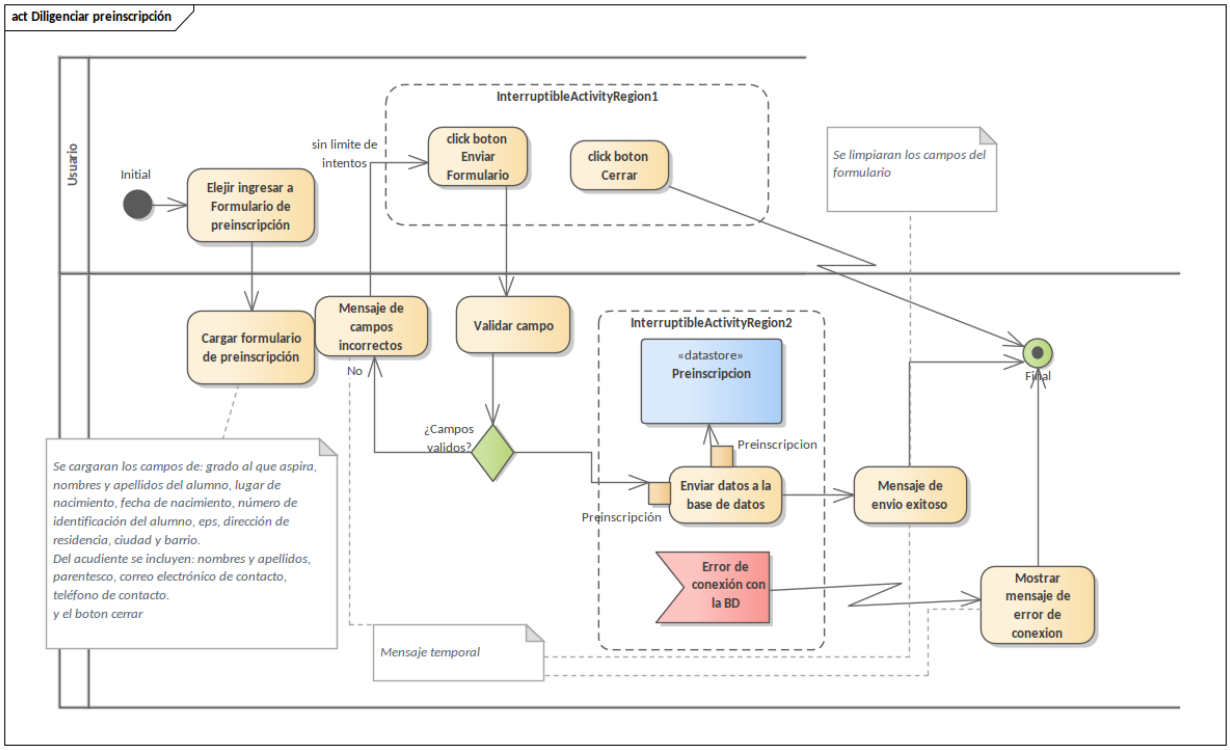
5.2 Formato extendido para casos de uso

5.2.1 Modulo Inicio de aplicación

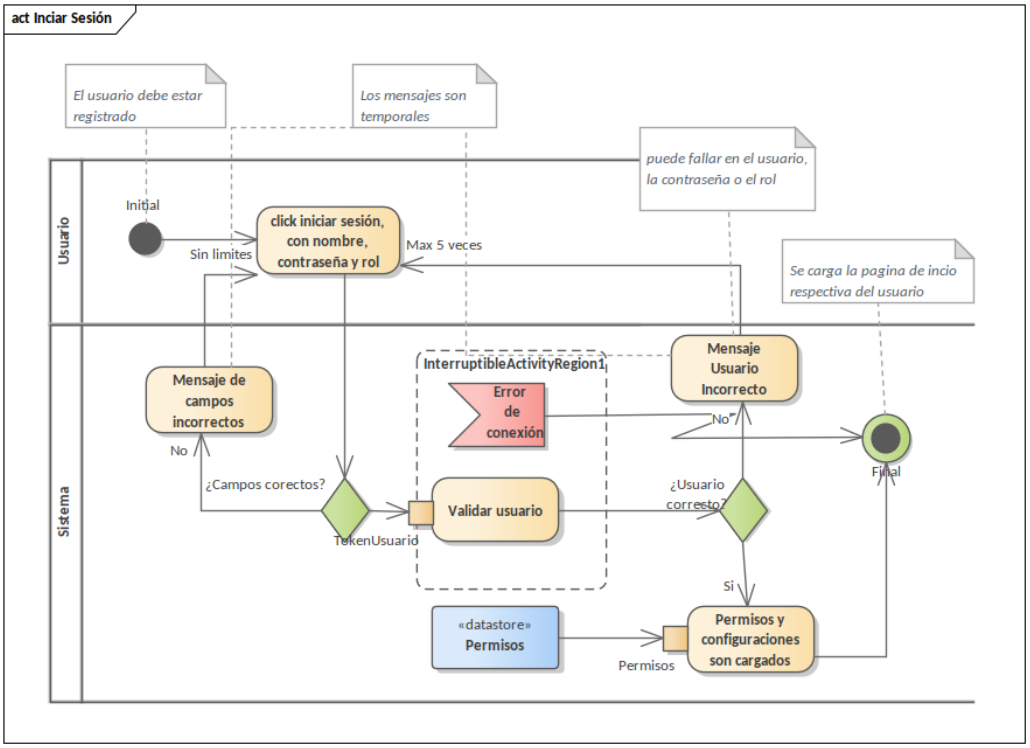
Caso de uso	**Iniciar aplicación**
Actores	Usuario
Descripción	Inicia la aplicación para empezar a usarla
Prioridad	Alta
Precondiciones	Ninguna
Postcondiciones	El sistema cargará la pantalla de inicio con las opciones de **formulario de preinscripción** e **iniciar sesión**

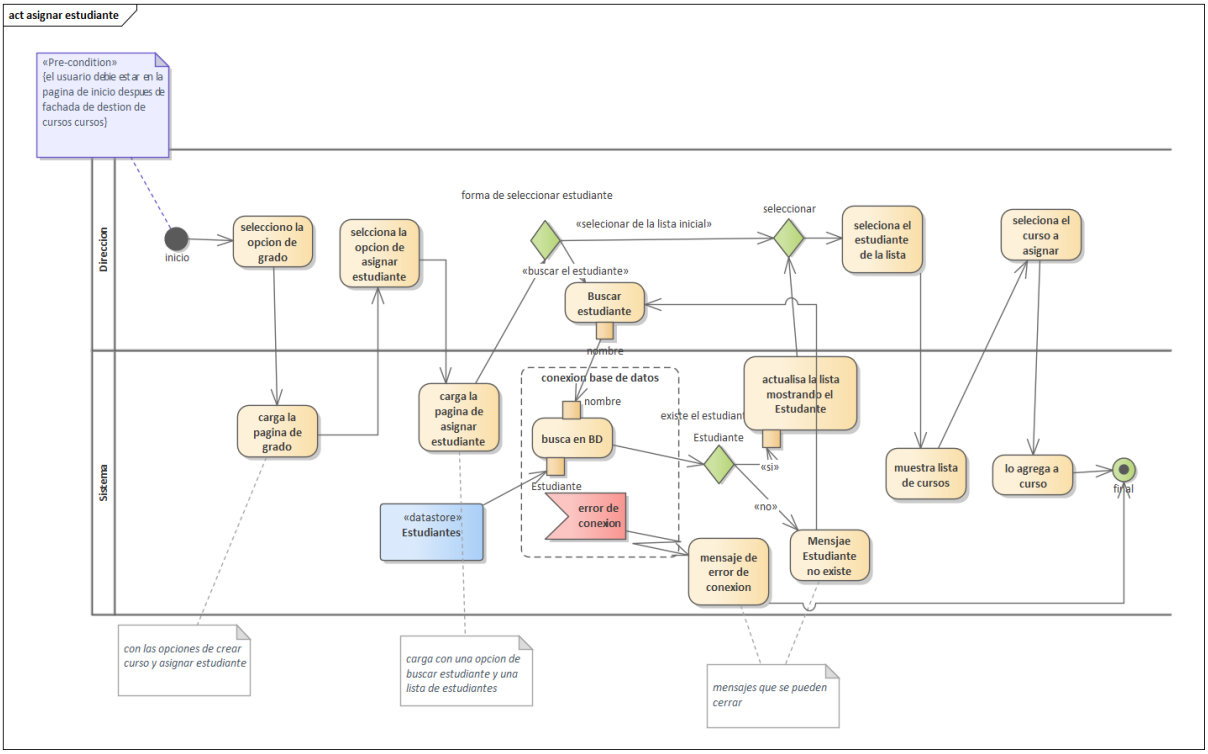


Caso de uso	**Diligenciar preinscripción.**
Actores	Usuario.
Descripción	La aplicación muestra un formulario que el acudiente del niño que aspira a ser parte de la institución llenará con información básica, incluyendo: Fecha en que se diligencia el cuestionario, año escolar en el que ingresaría el niño, grado al que aspira, nombres y apellidos del alumno, lugar de nacimiento, fecha de nacimiento, número de identificación del alumno, eps, dirección de residencia, ciudad y barrio. Del acudiente se incluyen: nombres y apellidos, parentesco, correo electrónico de contacto, teléfono de contacto, edad y ocupación.
Prioridad	Alta.
Precondiciones	-Haber iniciado la aplicación
Postcondiciones	Envía los datos llenados en el formulario a la base de datos.

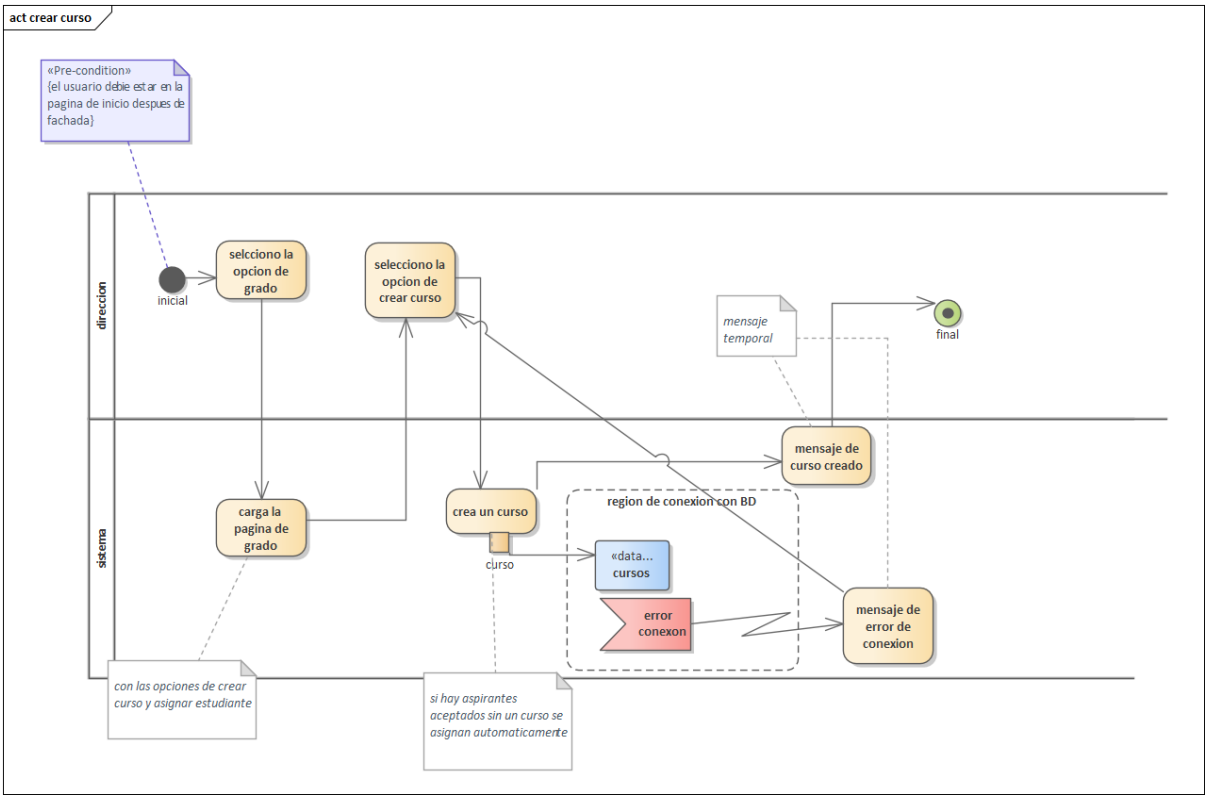


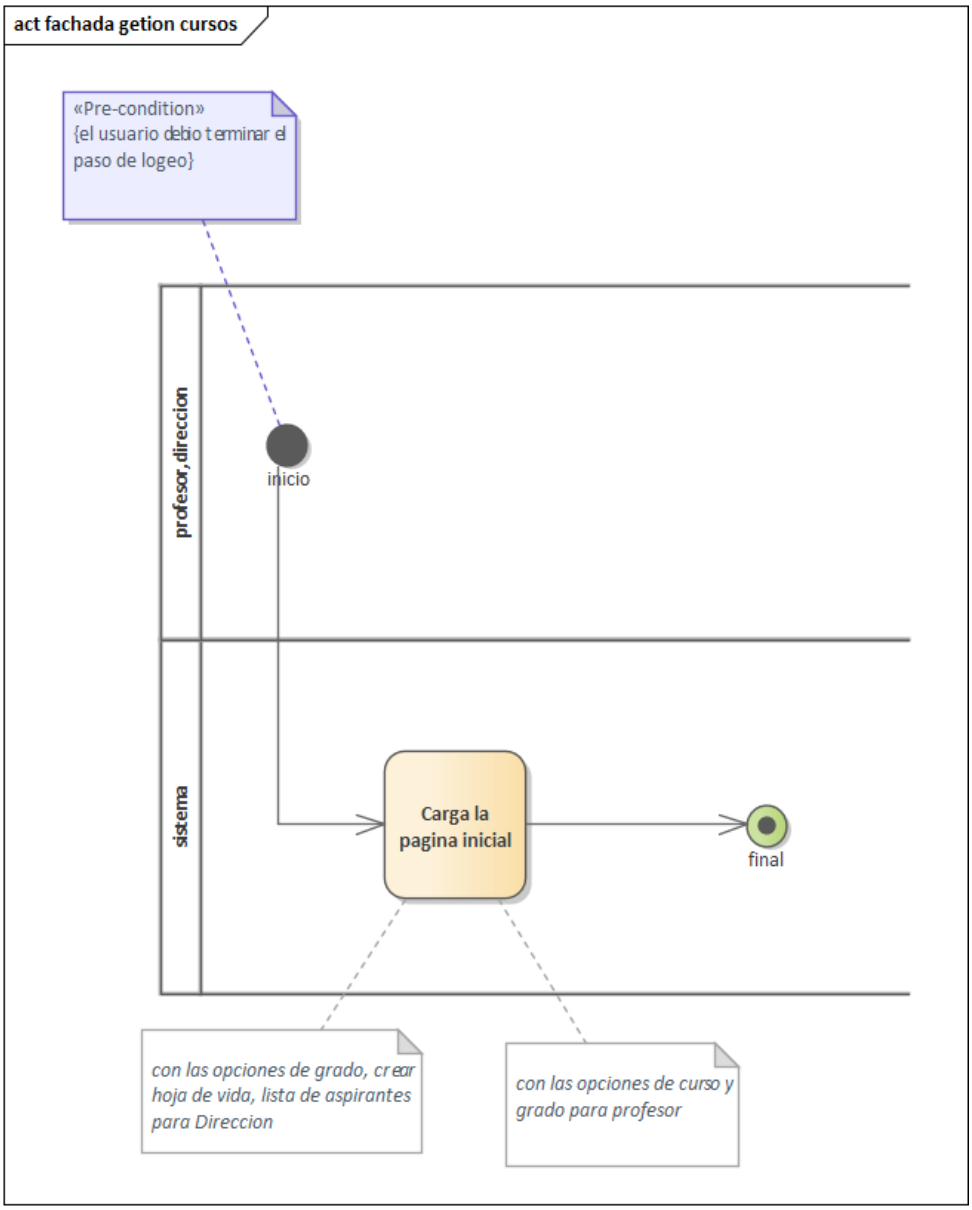
Caso de uso	**Iniciar sesión**
Actores	Usuario
Descripción	El usuario puede iniciar sesión para acceder a las funcionalidades del sistema según su rol
Prioridad	Alta
Precondiciones	-Haber iniciado la aplicación
Postcondiciones	-Inicia la sesión del usuario con las funcionalidades respectivas de su rol


act Inciar Sesión
El usuario debe estar registrado
Los mensajes son temporales
puede fallar en el usuario, la contraseña o el rol
Usuario
Sistema
Initial
Sin limites
click iniciar sesión, con nombre, contraseña y rol
Max 5 veces
Se carga la pagina de incio respectiva del usuario
Mensaje de campos incorrectos
No
¿Campos corectos?
TokenUsuarid
InterruptibleActivityRegion1
Error de conexión
Validar usuario
Mensaje Usuario Incorrecto
No
¿Usuario correcto?
Si
Final
«datastore» Permisos
Permisos
Permisos y configuraciones son cargados

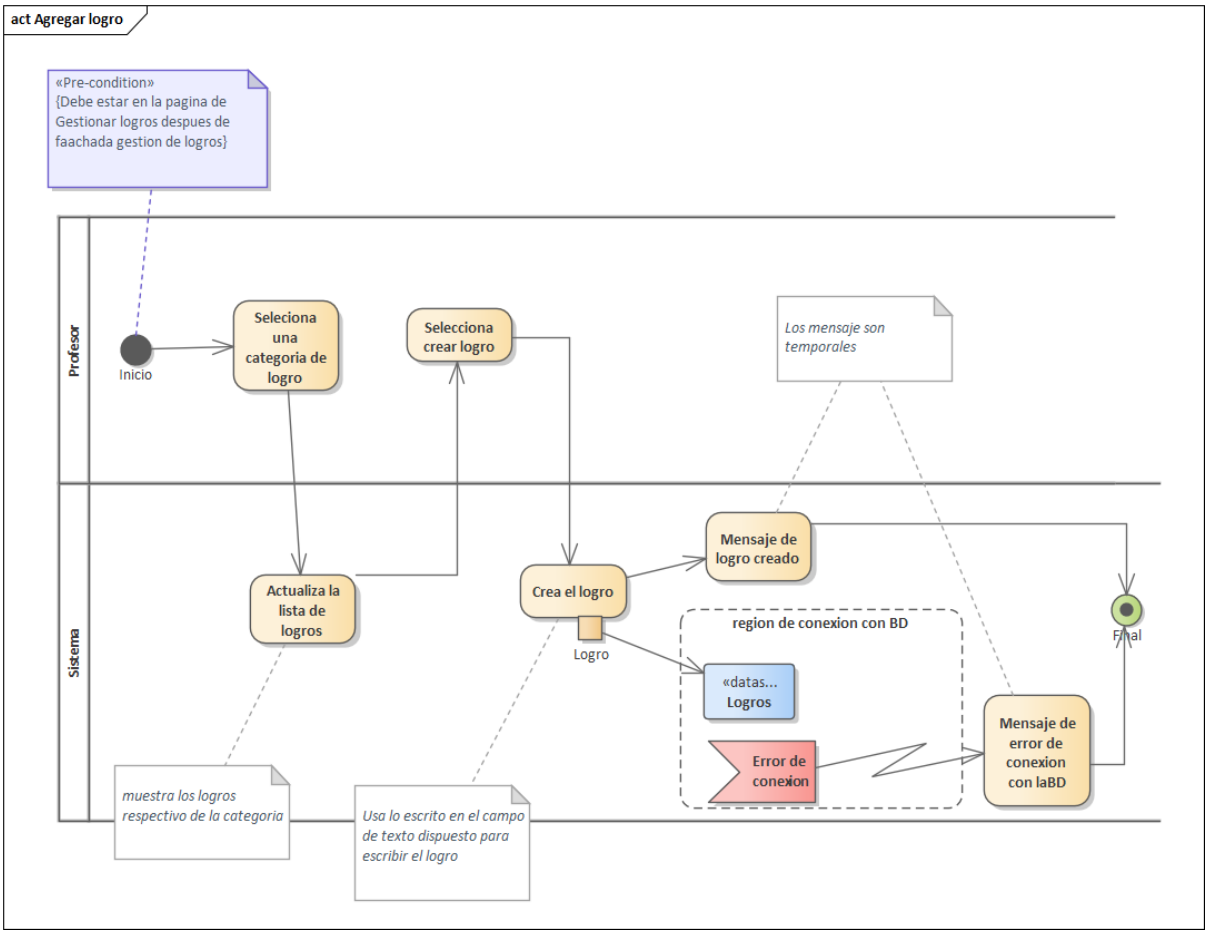

act asignar estudiante
«Pre-condition» {el usuario debe estar en la pagina de inicio despues de fachada de destion de cursos cursos}
Direccion
Sistema
inicio
selecciono la opcion de grado
selcciona la opcion de asignar estudiante
forma de seleccionar estudiante
«selecionar de la lista inicial»
«buscar el estudiante»
seleccionar
seleciona el estudiante de la lista
seleciona el curso a asignar
Buscar estudiante
nombre
carga la pagina de grado
carga la pagina de asignar estudiante
conexion base de datos
nombre
busca en BD
Estudiante
existe el estudiante
Estudiante
«si»
«no»
actualisa la lista mostrando el Estudante
muestra lista de cursos
lo agrega a curso
final
error de conexion
«datastore» Estudiantes
mensaje de error de conexion
Mensjae Estudiante no existe
con las opciones de crear curso y asignar estudiante
carga con una opcion de buscar estudiante y una lista de estudiantes
mensajes que se pueden cerrar

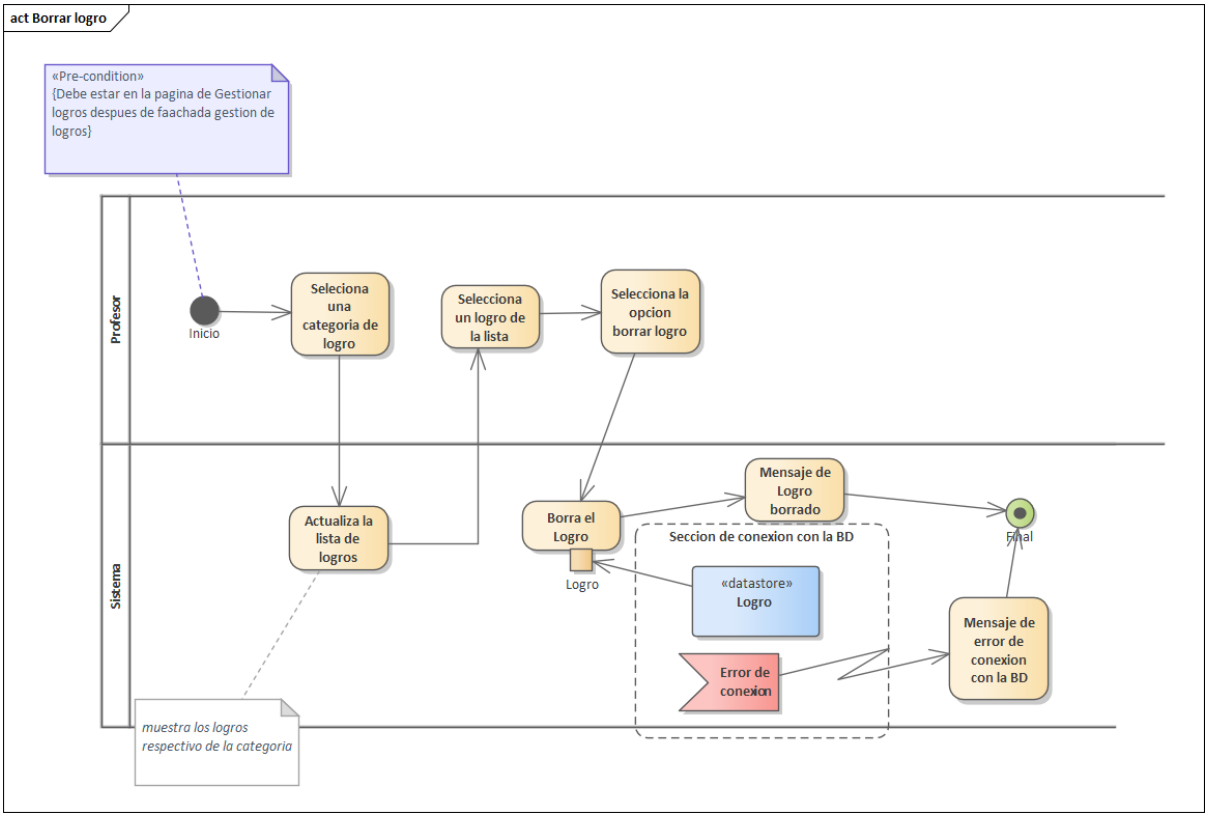
5.2.2 Modulo Gestión de cursos




act fachada getion cursos
«Pre-condition»
{el usuario debio terminar el paso de logeo}
profesor, direccion
sistema
inicio
Carga la pagina inicial
final
con las opciones de grado, crear hoja de vida, lista de aspirantes para Direccion
con las opciones de curso y grado para profesor

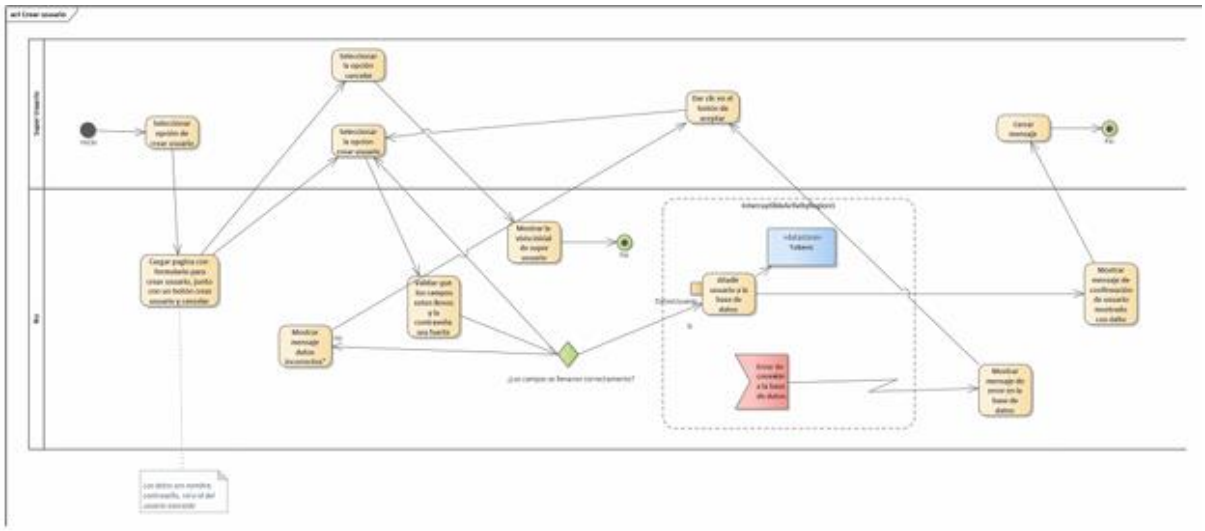
5.2.3 Modulo Gestión de logros

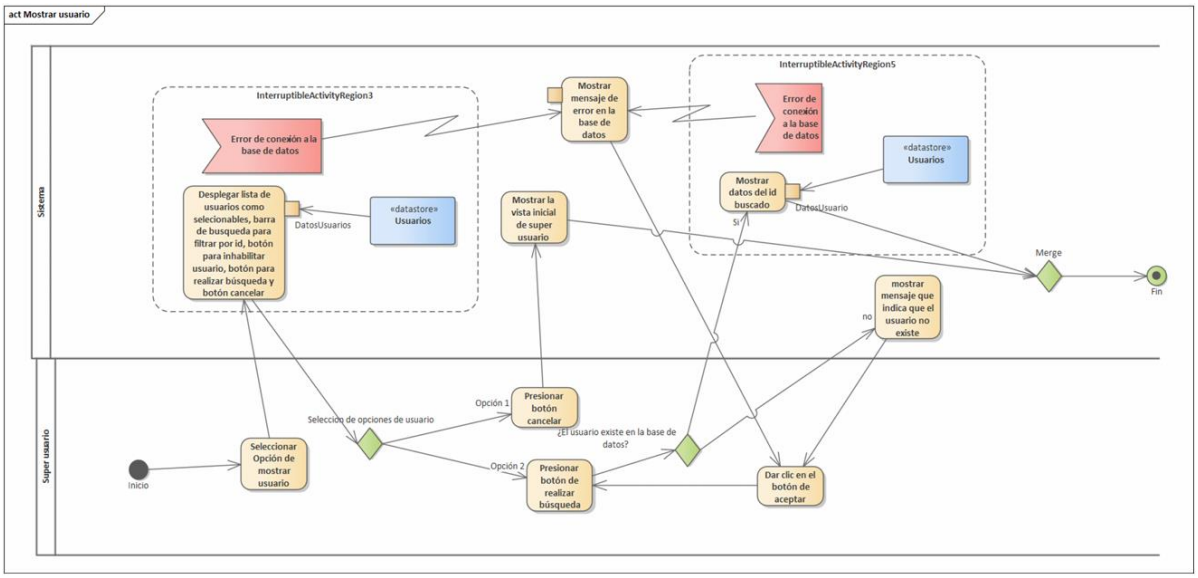


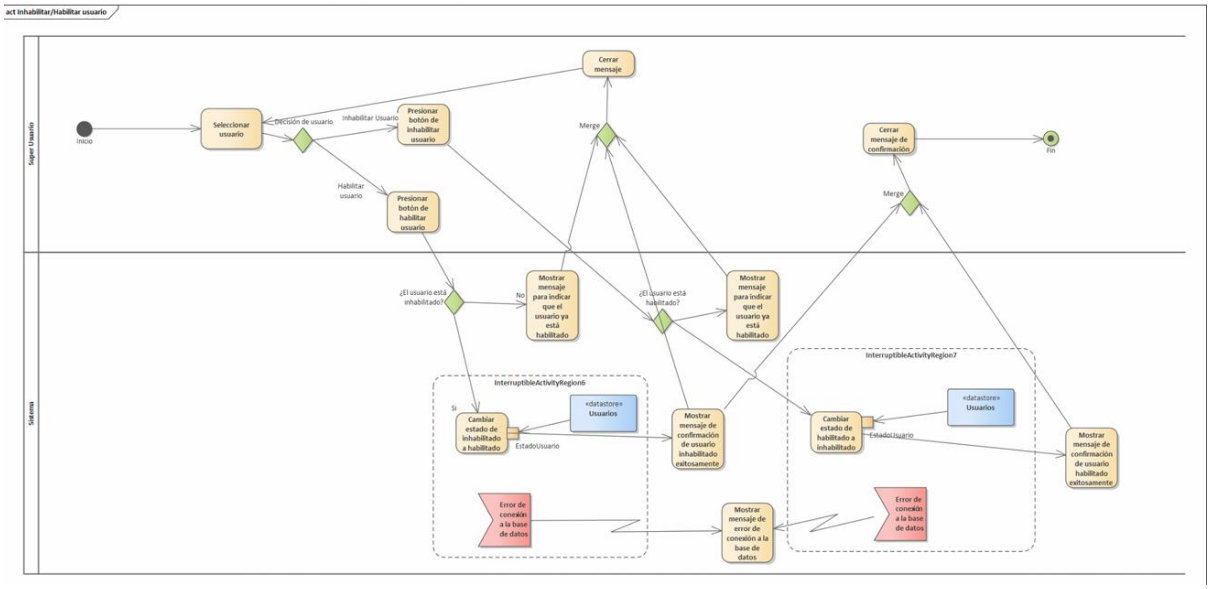


5.2.4 Modulo Gestión de Estudiantes

5.2.5 Modulo Gestión de usuarios





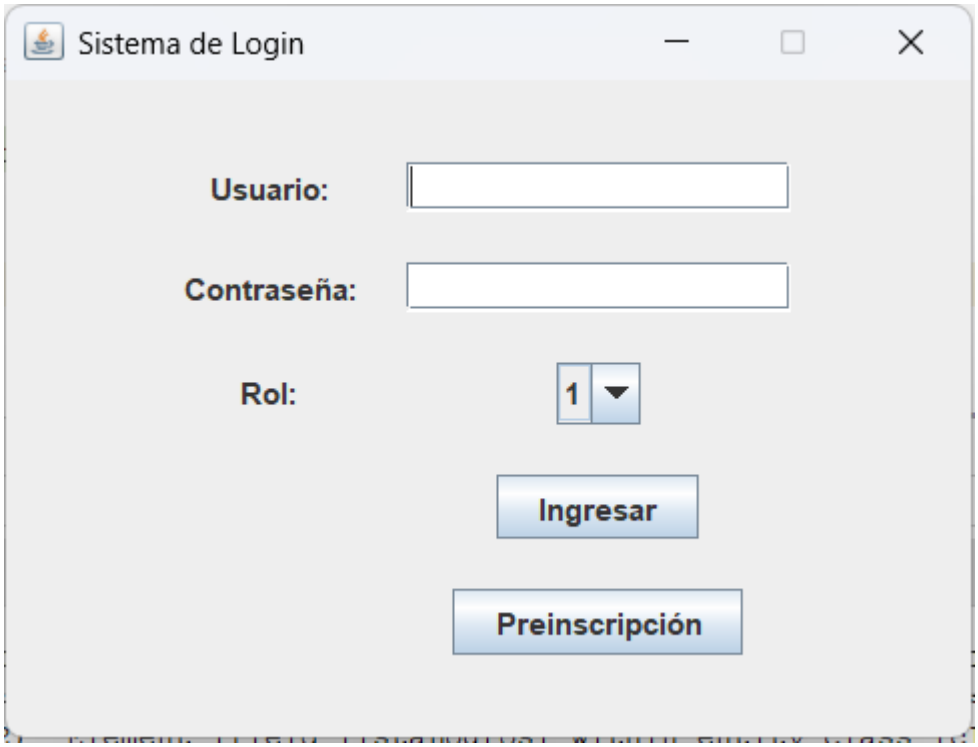


6. BOCETOS VISUALES DE INTERFAZ

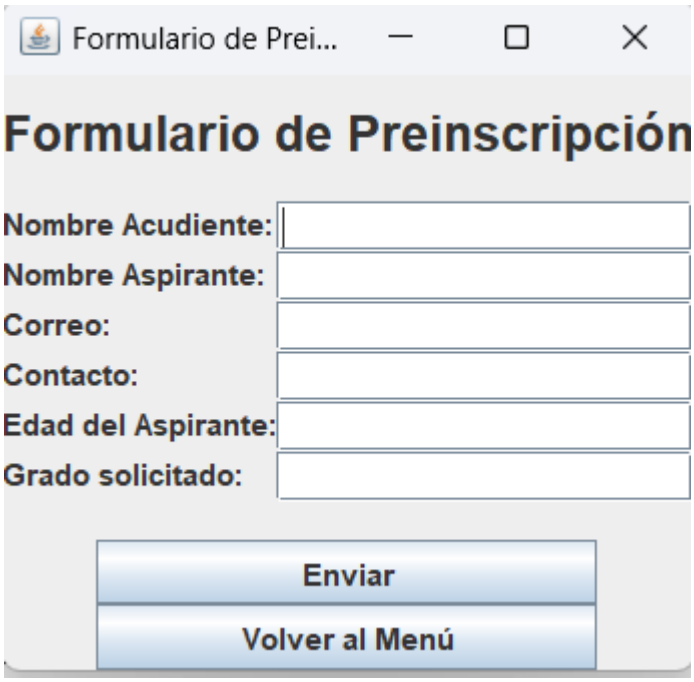
Las siguientes imágenes son bocetos primarios de cómo se verá las distintas vistas e interfaces de la aplicación, tener en cuenta que pueden surgir modificaciones o tener cierta diferencia con el prototipo inicial de acuerdo con los casos de usos aceptados en el acuerdo, los bocetos estarán en el orden de usuario: Usuario, Superusuario, Acudiente, Dirección y Profesor

6.1 Vista Usuario

Inicio de aplicación

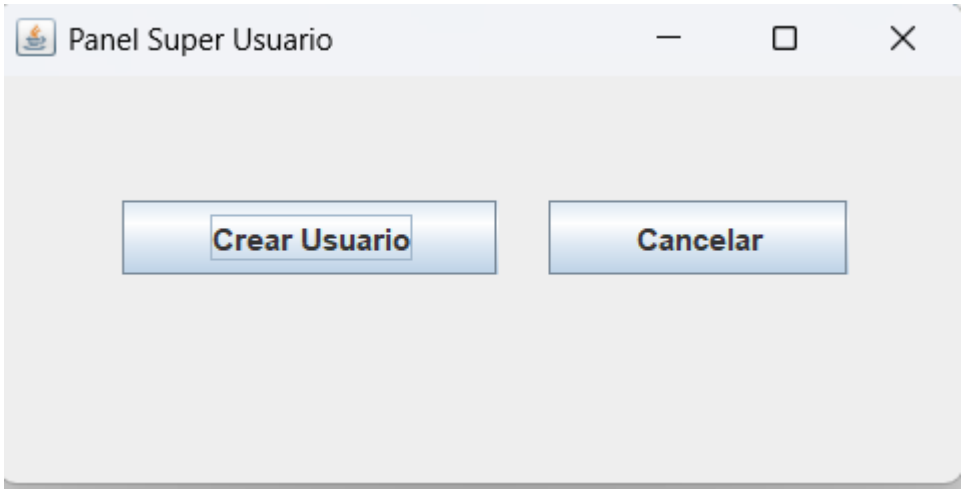


Formulario de preinscripción

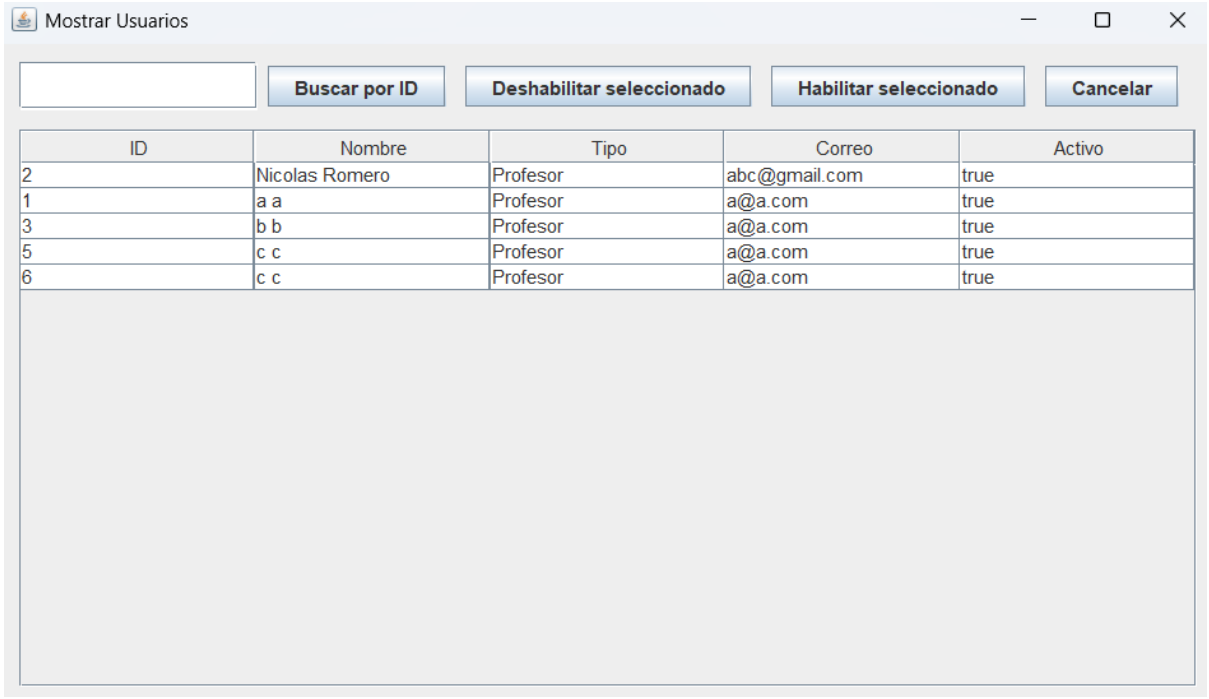


6.2 Vista de Superusuario

inicio

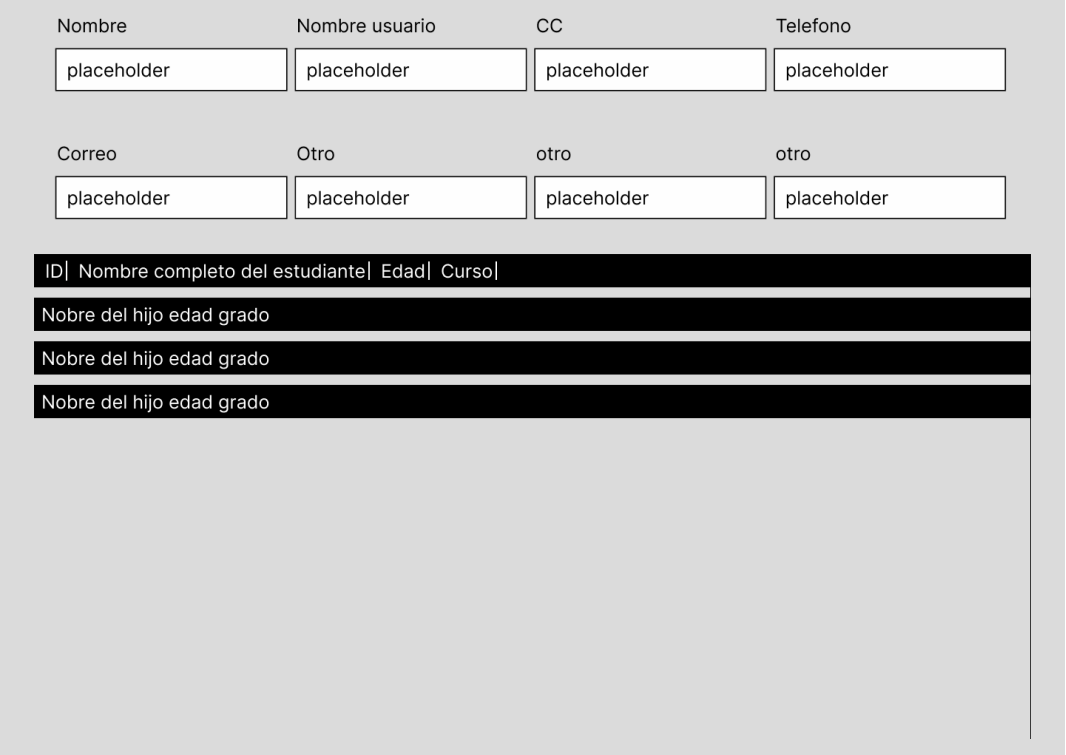


Crear usuarios

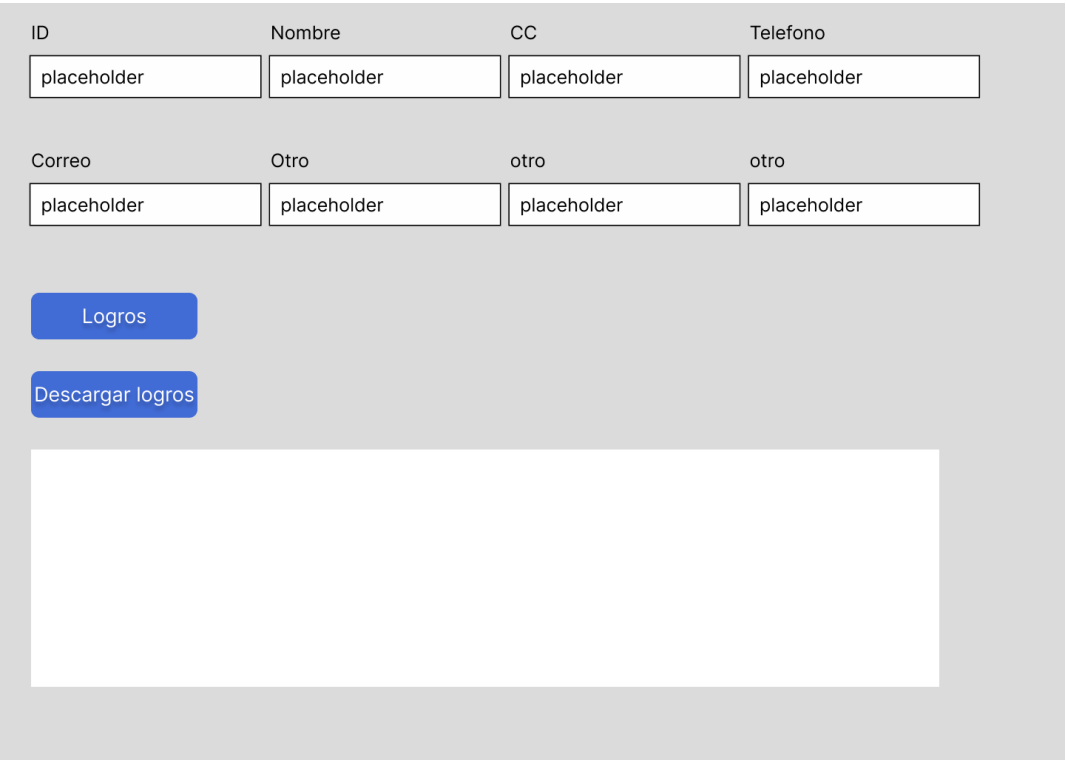


6.3 Vista de Acudiente

Inicio



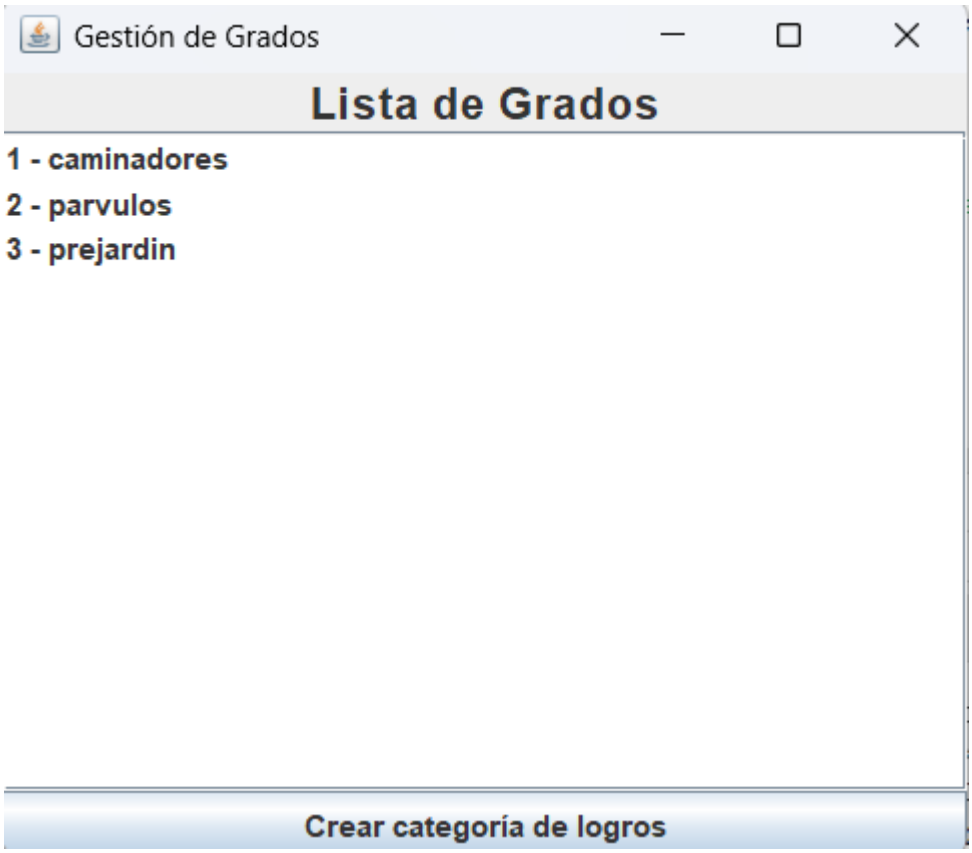
Perfil de estudiante



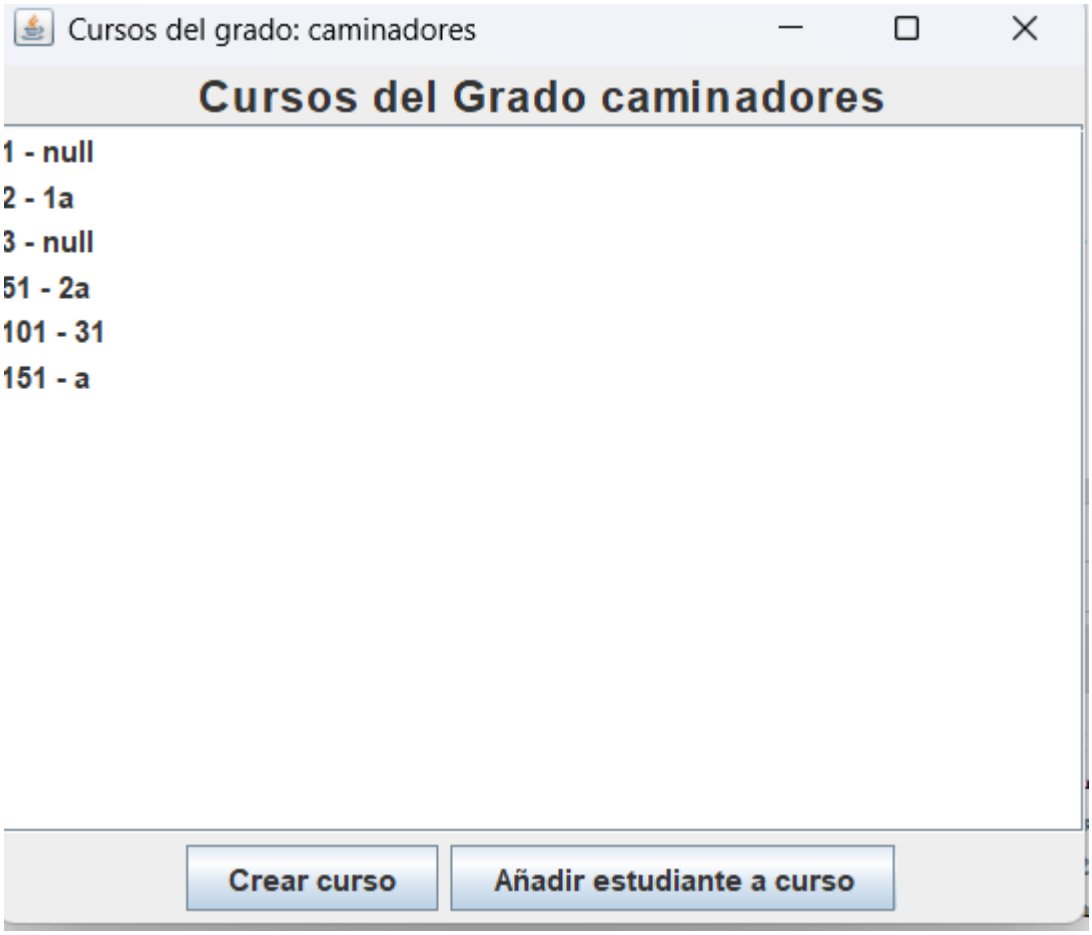
6.4 Vista de Dirección

Inicio

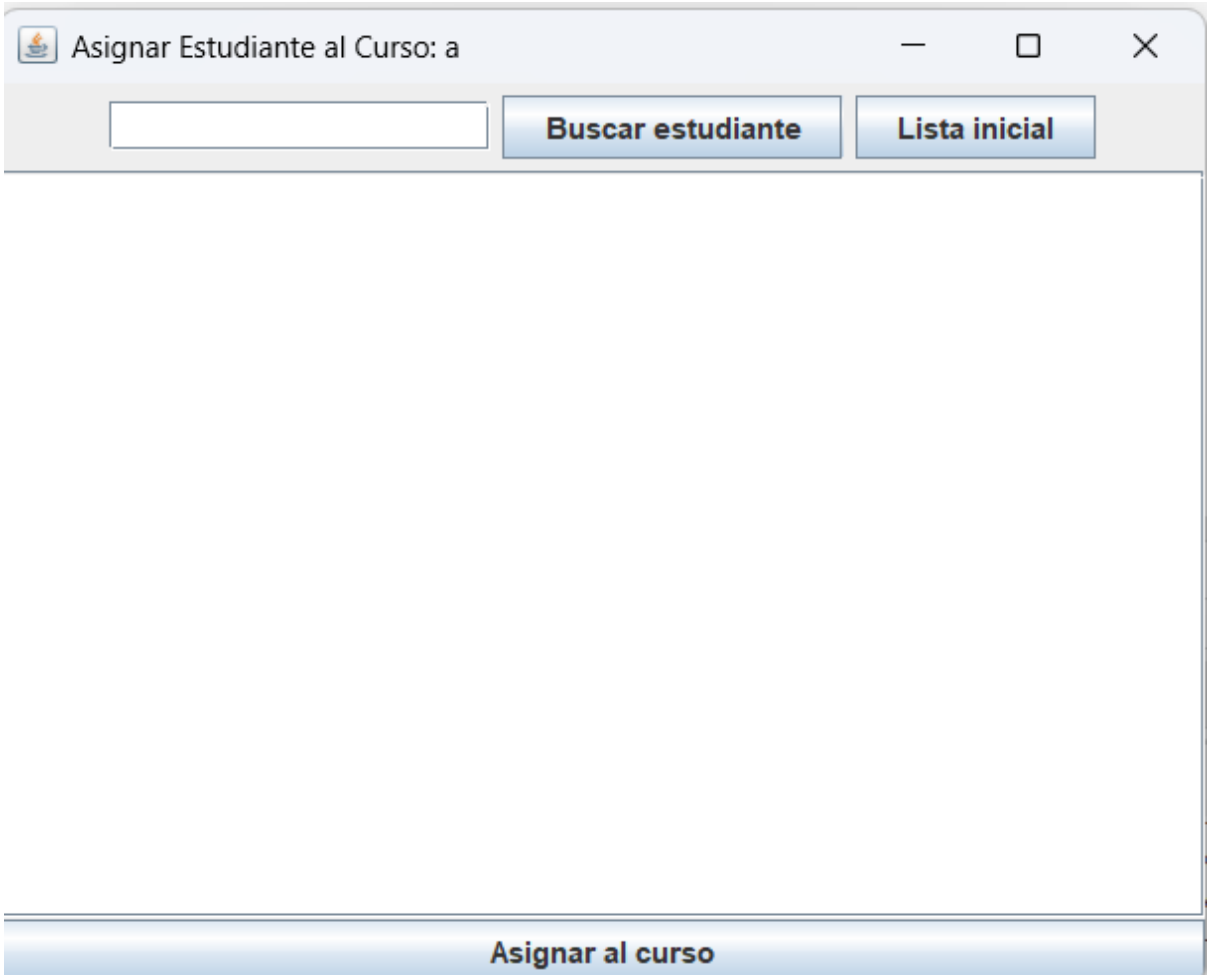
Grado



Asignar Estudiante

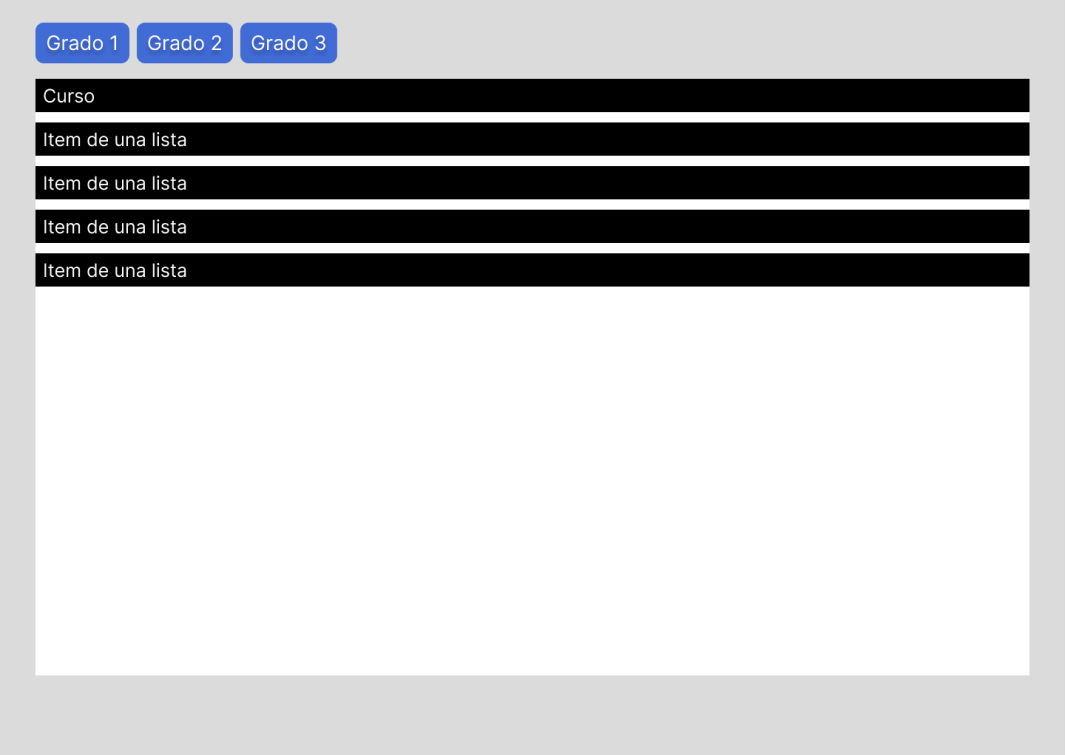


Añadir Estudiante

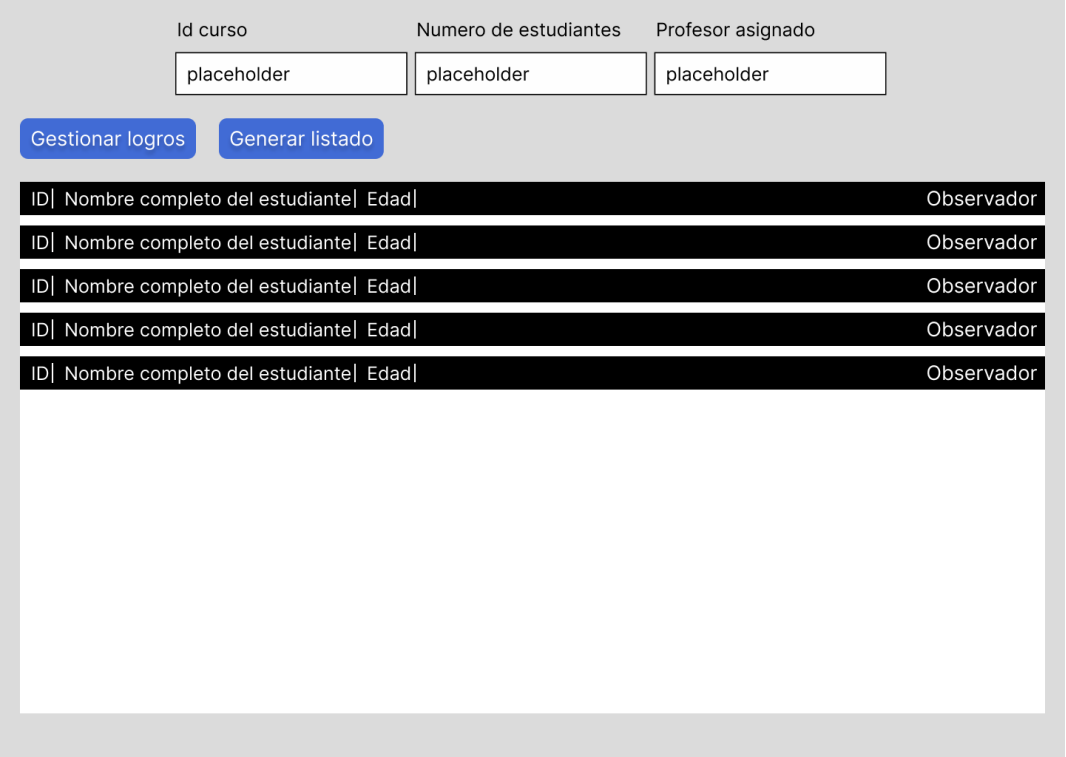


6.5 Vista de profesor

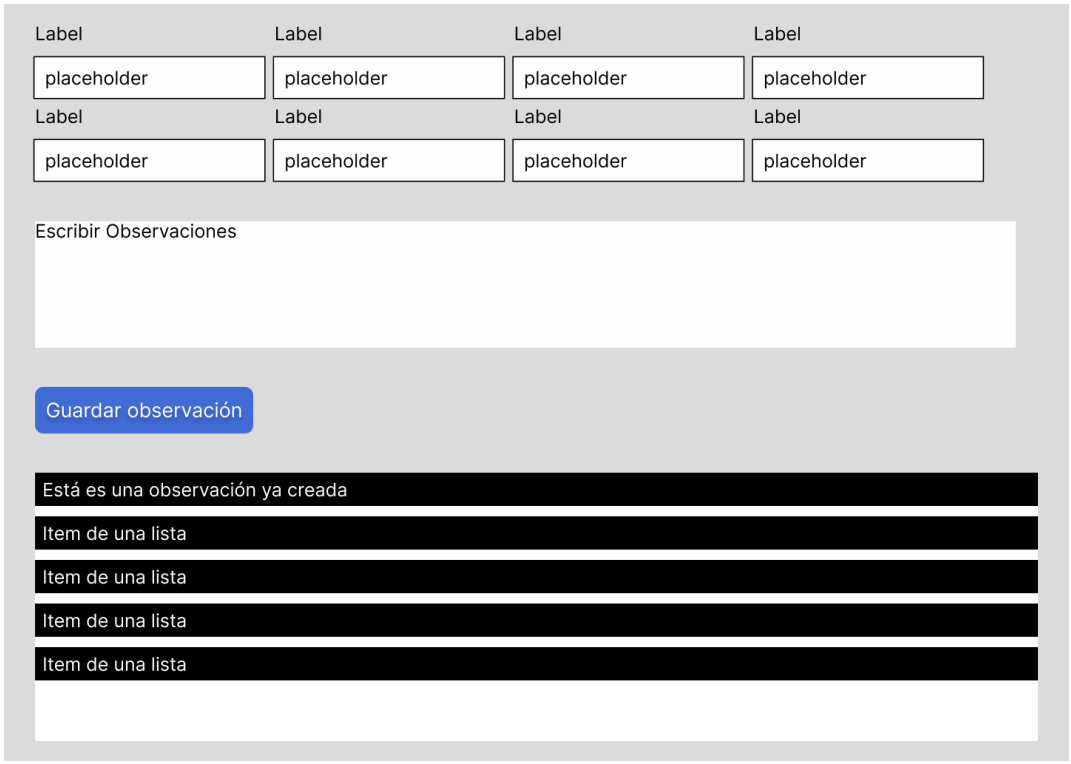
Inicio



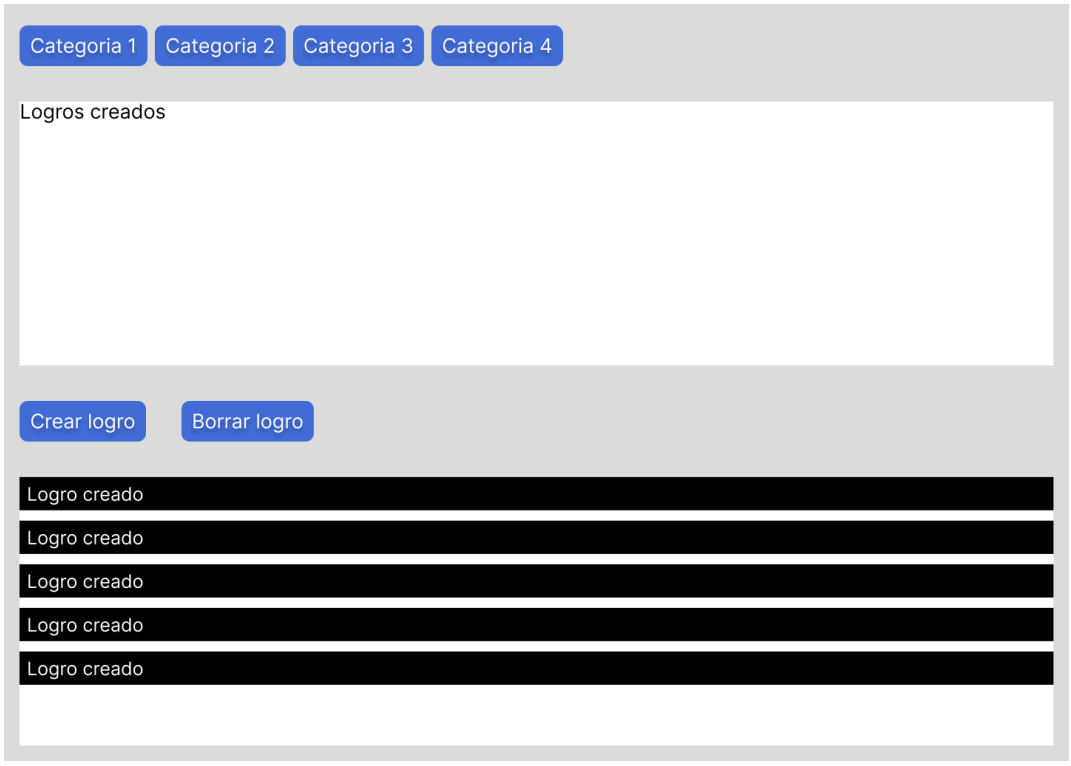
Curso



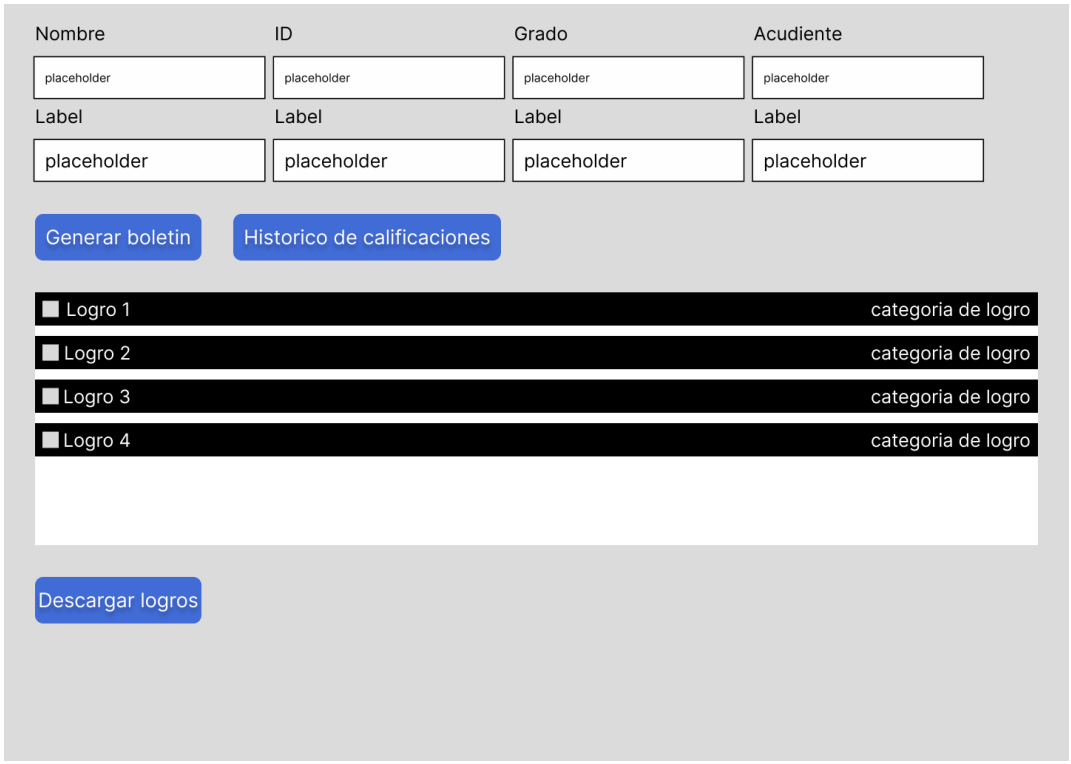
Observador



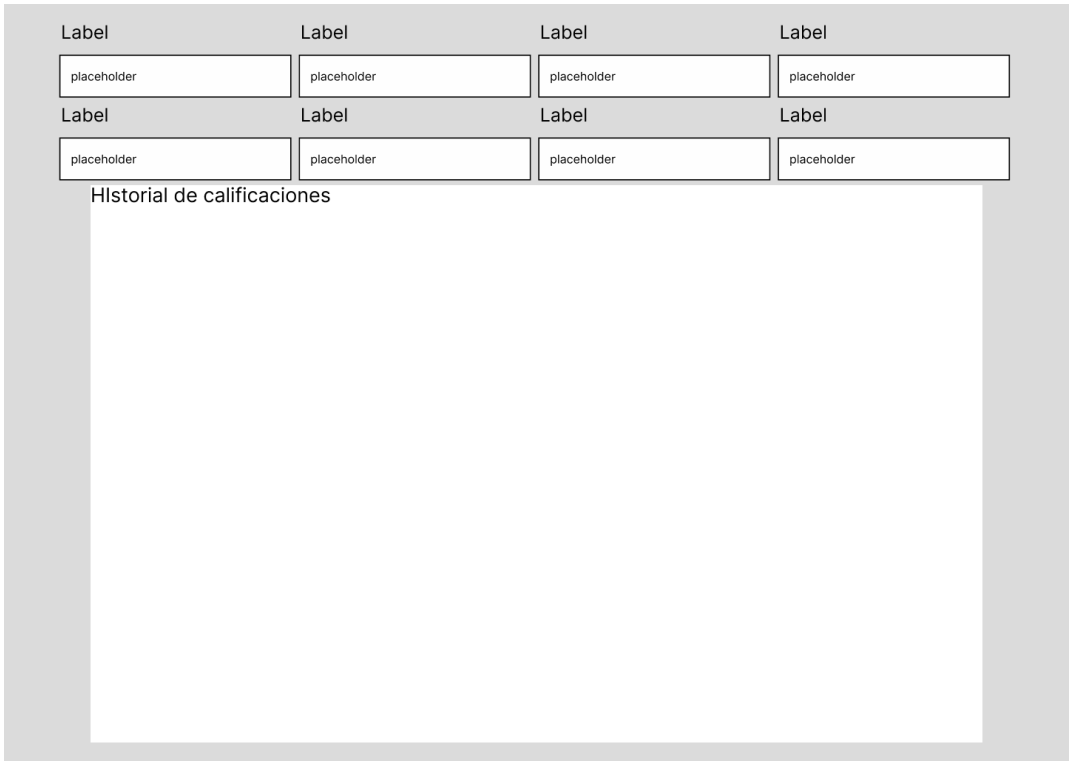
Crear Logros



Perfil estudiante



Historial de calificaciones



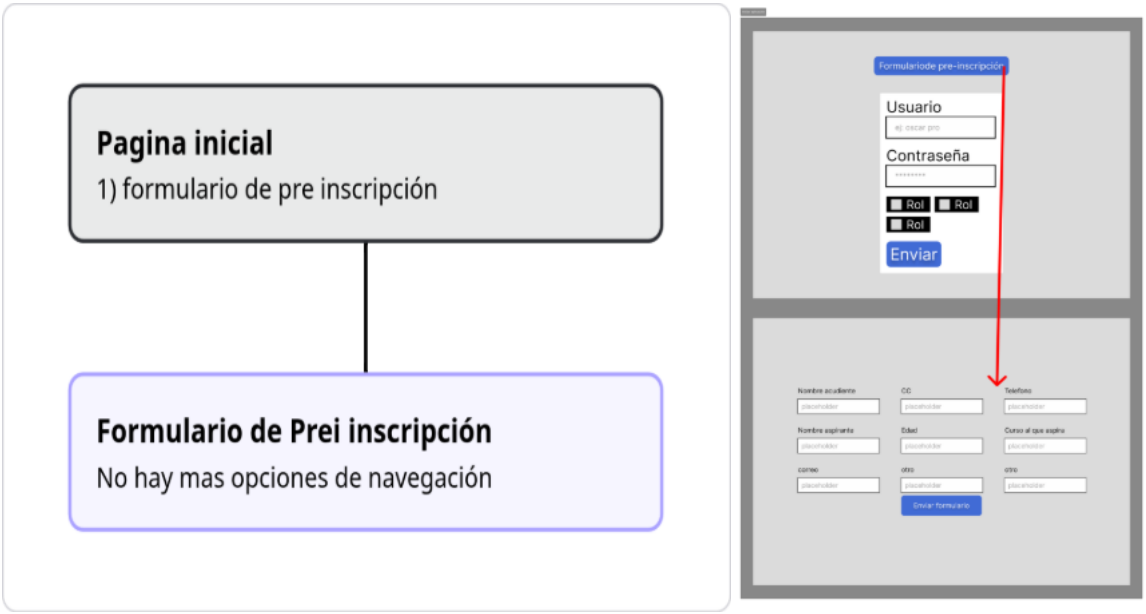
CAPITULO 2 MODELO ESTRUCTURAL

2. MAPA DE NAVEGACION

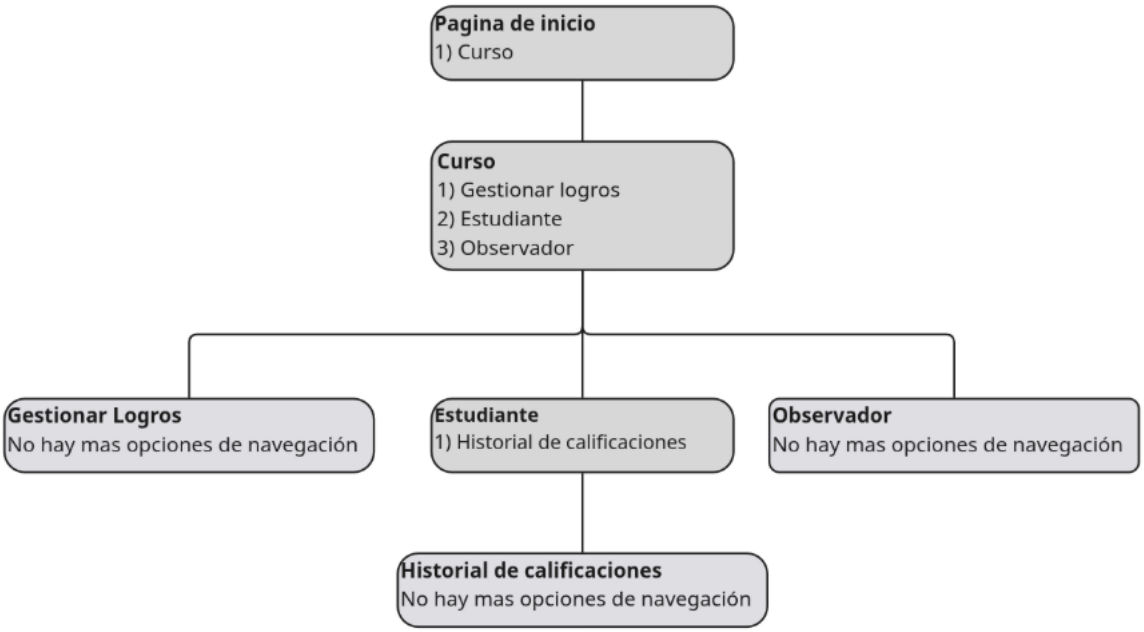
Para la sección de navegación se guiará de dos imágenes una que son las **opciones de navegación** y la otra el **flujo de navegación** en el boceto, en la primera estará en negro el “nombre” de la página y debajo enumeradas las opciones de navegación, y en la de boceto se guía por color, donde rojo, son las primeras opciones, violeta las segundas opciones, y en verde las terceras opciones.

Para una mayor claridad se agregarán las imágenes de flujo sobre el boceto, en orden con mayor definición al final del documento

2.1 Vista inicial de la aplicación



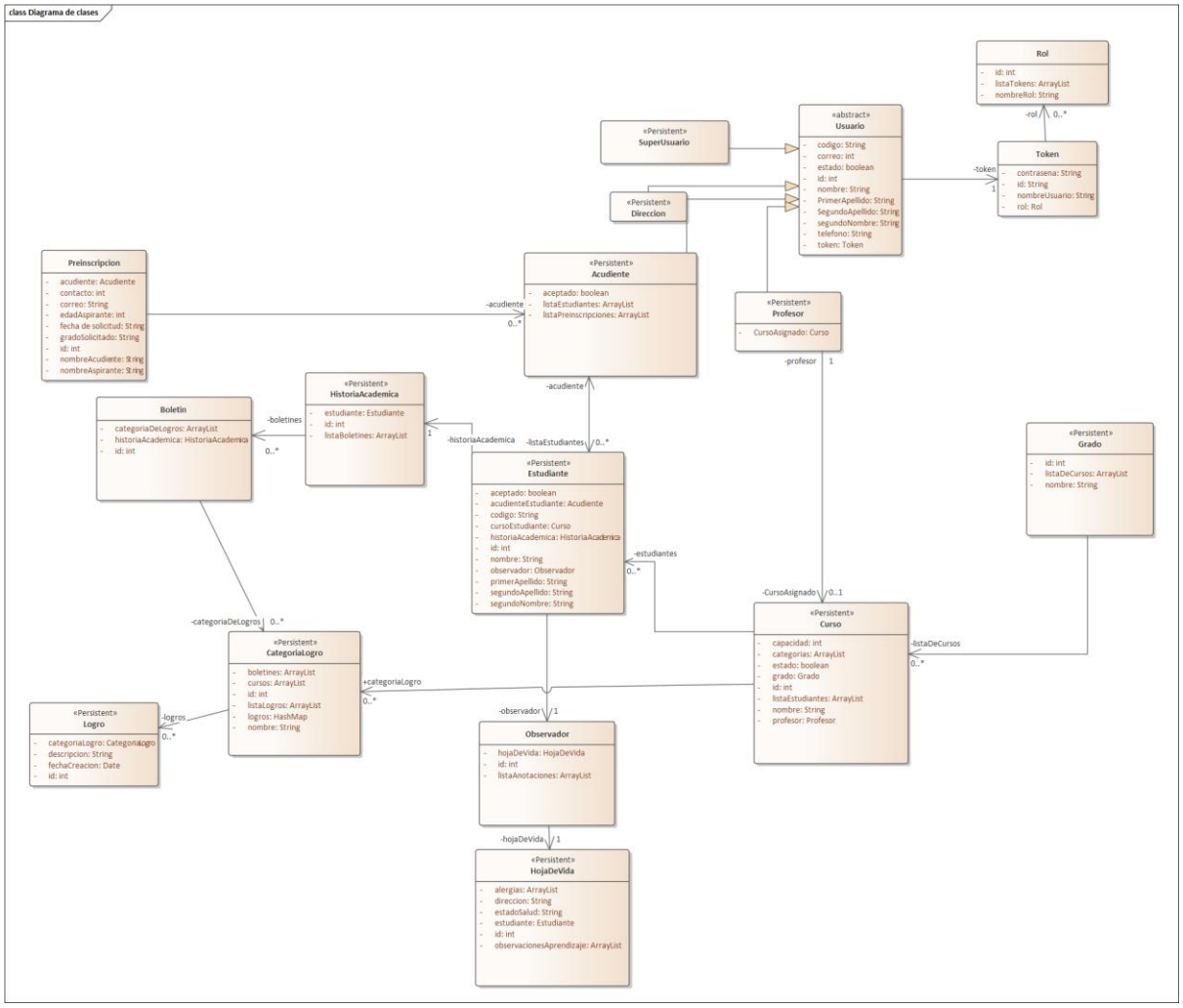
2.2 Vista profesor



3. MODELO DOMINIO

En esta sección se presenta el diagrama de clases, el cual representa el modelo de dominio del problema. Cada clase está compuesta por una capa de atributos, que describen las propiedades y características de las entidades del sistema, y una capa de métodos, que definen las operaciones o comportamientos que dichas clases ejecutarán una vez se desarrolle el código de la aplicación. Asimismo, se muestran las relaciones existentes entre las clases propuestas, junto con su correspondiente navegabilidad y multiplicidad, lo que permite comprender cómo interactúan los distintos componentes del sistema.

Este diagrama constituye la base conceptual del diseño orientado a objetos, ya que permite visualizar la estructura lógica del sistema antes de su implementación. Además, sirve como punto de partida para la transformación al modelo relacional, garantizando que la estructura de datos que se implementará en la base de datos mantenga la coherencia con la lógica del dominio académico, en el cual intervienen entidades como estudiantes, profesores, cursos y grados.



4. DICIONARIO DE CLASES

Esta sección presenta un componente esencial del modelo de dominio del sistema. Su propósito es establecer una **terminología estandarizada y consistente** para todas las clases identificadas durante la fase de análisis

A continuación, se dará una **definición textual y detallada** de cada una de las **clases** identificadas, con atributos y sus métodos, asegurando un **significado común y consistente**. Todas las clases cuentan con sus respectivos **getters** y **setters**

Rol				
Atributos				
Visibilidad	Nombre	Tipo	Multiplicidad	Dominio de valores

private	id	int		1-2000000
private	nombreRol	String		[a-z,A-Z]
private	permiso	Permisos	1	Objeto de tipo Permisos

Token				
Atributos				
Visibilidad	Nombre	Tipo	Multiplicidad	Dominio de valores
private	usuario	String		[a-z],[A-Z]
private	contraseña	String		
private	rol	Rol	1.*	Objeto de tipo Rol

Usuario <abstract>				
Atributos				
Visibilidad	Nombre	Tipo	Multiplicidad	Dominio de valores
private	id	int		1-2000000
private	código	String		[0-9]
private	nombre	String		[a-z,A-Z]
private	segundoNombre	String		[a-z,A-Z]
private	primerNombre	String		[a-z,A-Z]
private	segundoApellido	String		[a-z,A-Z]
private	estado	boolean		false, true
private	correo	String		[a-z,@,., 0-9]
private	teléfono	String		[0-9]

private	rol	Rol	1	Objeto de tipo Rol

Profesor::Usuario				
Atributos				
Visibilidad	Nombre	Tipo	Multiplicidad	Dominio de valores
private	cursoAsignado	Curso	1	Objeto de tipo curso

Acudiente::Usuario				
Atributos				
Visibilidad	Nombre	Tipo	Multiplicidad	Dominio de valores
private	ListaEstudiantes	ArrayList	1.*	Lista de Objetos de tipo Estudiante
private	aceptado	boolean		true, false
private	preinscripciones	Preinscripcion	1.*	Lista de Objetos de tipo Preinscripcion

Direccion::Usuario				
Atributos				
Visibilidad	Nombre	Tipo	Multiplicidad	Dominio de valores

Super usuario::Usuario
Atributos

Visibilidad	Nombre	Tipo	Multiplicidad	Dominio de valores

Estudiantes				
Atributos				
Visibilidad	Nombre	Tipo	Multiplicidad	Dominio de valores
private	id	int		1-2000000
private	nombre	String		[a-z,A-Z]
private	observador	Observador	1	Objeto de tipo observador
private	aceptado	boolean		true, false
private	codigo	String		[0-9]
private	acudiente	Acudiente	1	Objeto de tipo acudiente
private	historiAcademica	HistoriaAcademica	1	Objeto de tipo HistoriaAcademica
private	segundoNombre	String		[a-z,A-Z]
private	primerApellido	String		[a-z,A-Z]
private	segundoApellido	String		[a-z,A-Z]

Grado				
Atributos				
Visibilidad	**Nombre**	**Tipo**	**Multiplicidad**	**Dominio de valores**
private	id	int		1-2000000
private	nombre	String		[a-z,A-Z,0-9]
private	listaCursos	ArrayList	0.*	Lista de Objetos de tipo cursos

Curso				
Atributos				
Visibilidad	Nombre	Tipo	Multiplicidad	Dominio de valores
private	id	int		1-2000000
private	nombre	String		[a-z,A-Z]
private	estudiantes	HashMap	1.*	Lista de Objetos de tipo estudiantes
private	capacidad	int		0-20
private	profesor	Profesor	1	Objetos de tipo Profesor
private	grado	Grado	1	Objetos de tipo Grado
private	categoriaLogros	ArrayList	1.*	Lista de Objetos de tipo categotiaLogro
private	estado	boolean		true, false
Métodos				
Visibilidad	Nombre	Parámetro	Tipo	Intencionalidad
public	recuperarListaEstudiantes		HashMap	Devuelve la lista de estudiantes
public	esValido		boolean	Validar si el curso tiene almenos 5 estudiantes para activarse
public	agregarCategoria	CategoriaLogro	void	Agregar un Objeto de categoriaLogro a la lista categoriaLogros

public	eliminarCategoria	CategoriaLogro	void	Eliminar un Objeto de categoriaLogro a la lista categoriaLogros
public	agregarEstudiante	Estudiante	void	Agregar un Objeto de Estudiante a la lista estudiantes
public	buscarCategoria	String	void	Buscar un Objeto de CategoriaLogro en la lista categoriaLogros

pre-inscripción				
Atributos				
Visibilidad	Nombre	Tipo	Multiplicidad	Dominio de valores
private	nombreAcudiente	String		[a-z.A-Z]
private	telefono	string		[0-9]
private	correo	String		[a-z,A-Z.@,.,0-9]
private	fechaSolicitud	String		[0-9,/]
private	nombreAspirante	String		[a-z,A-Z]
private	gradoSolicitado	String		[a-z,A-Z]
private	contacto	String		[0-9]

Observador
Atributos

Visibilidad	Nombre	Tipo	Multiplicidad	Dominio de valores
Private	anotaciones	Array		Lista de Strings [a-z,A-Z,0-9]
private	hojaDeVida	HojaDeVida	1	Objeto de tipo HojaDeVida
public	agregarAnotacion	String	void	Agrega una anotación a la lista anotaciones

HojaDeVida				
Atributos				
Visibilidad	Nombre	Tipo	Multiplicidad	Dominio de valores
Private	estudiante	Estudiante	1	Objeto de tipo Estudiante
Private	alergias	Array		Lista de Strings [a,z,A-Z,0-9]
Private	estadoSalud	String		[a-z,A-Z,0-9]
Private	capacidadesAprendizaje	Array		Lista de Strings [a,z,A-Z,0-9]
private	direccion	String		[0-9,-,#,a-z,A-Z]
Visibilidad	Nombre	Parámetro	Tipo	Intencionalidad
public	buscarObservacion	int	String	Busca un Objeto de la lista capacidadesAprendizaje
public	eliminarObservacion	int	void	Elimina un Objeto de la lista capacidadesAprendizaje
public	eliminarObservacion	int	void	Elimina un Objeto de la lista capacidadesAprendizaje
public	EliminarAlergia	String	void	Elimina un Objeto de la lista alergias

public	buscarAlergia	String	String	Busca un Objeto de la lista alergias
public	agregarAlergia	String	void	Agrega un Objeto a la lista alergias
public	agregarObservacion	String	void	Agrega un Objeto a la lista capacidadesAprendizaje

HistoriaAcademica				
Atributos				
Visibilidad	Nombre	Tipo	Multiplicidad	Dominio de valores
Private	estudiante	Estudiante	1	Objeto de tipo Estudiante
Private	boletines	Array		Lista de Objetos de tipo Boletines

Boletín				
Atributos				
Visibilidad	Nombre	Tipo	Multiplicidad	Dominio de valores
private	id	int		1-2000000
Private	categoriaLogros	Array	0.*	Lista de Objetos de tipo categoriasLogro

CategoriaLogros

Atributos				
Visibilidad	Nombre	Tipo	Multiplicidad	Dominio de valores
private	id	int		1-2000000
Private	nombre	String		[a-z,A-Z]
Private	logros	Array	0.*	Lista de Objetos de tipo Logros

Logro				
Atributos				
Visibilidad	Nombre	Tipo	Multiplicidad	Dominio de valores
private	id	int		1-2000000
Private	descripcion	String		[a-z,A-Z]
Private	fechaCreacion	String		[0-9,/]

5. MODELO DE PERSISTENCIA

5.1 Estrategias de mapeo

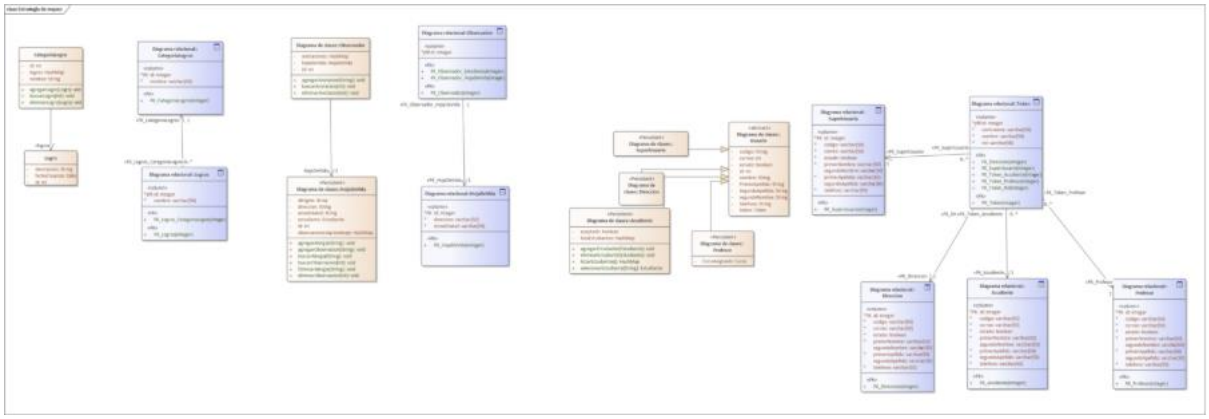
A continuación, se presentan las estrategias utilizadas para lograr que el modelo de persistencia representado por el diagrama de clases se pueda transformar en un modelo relacional sin generar pérdida de información.

Para manejar las herencias en el esquema relacional se hizo uso de mapeo a clase concreta, por lo que en este no se creó una tabla equivalente a la clase abstracta “Usuario” del modelo de dominio, en su lugar se optó por hacer tablas que correspondieran a “Acudiente”, “Dirección”, “Profesor” y “SuperUsuario” en donde

cada una de estas también contienen los atributos que la clase usuario tenía en el modelo orientado a objetos.

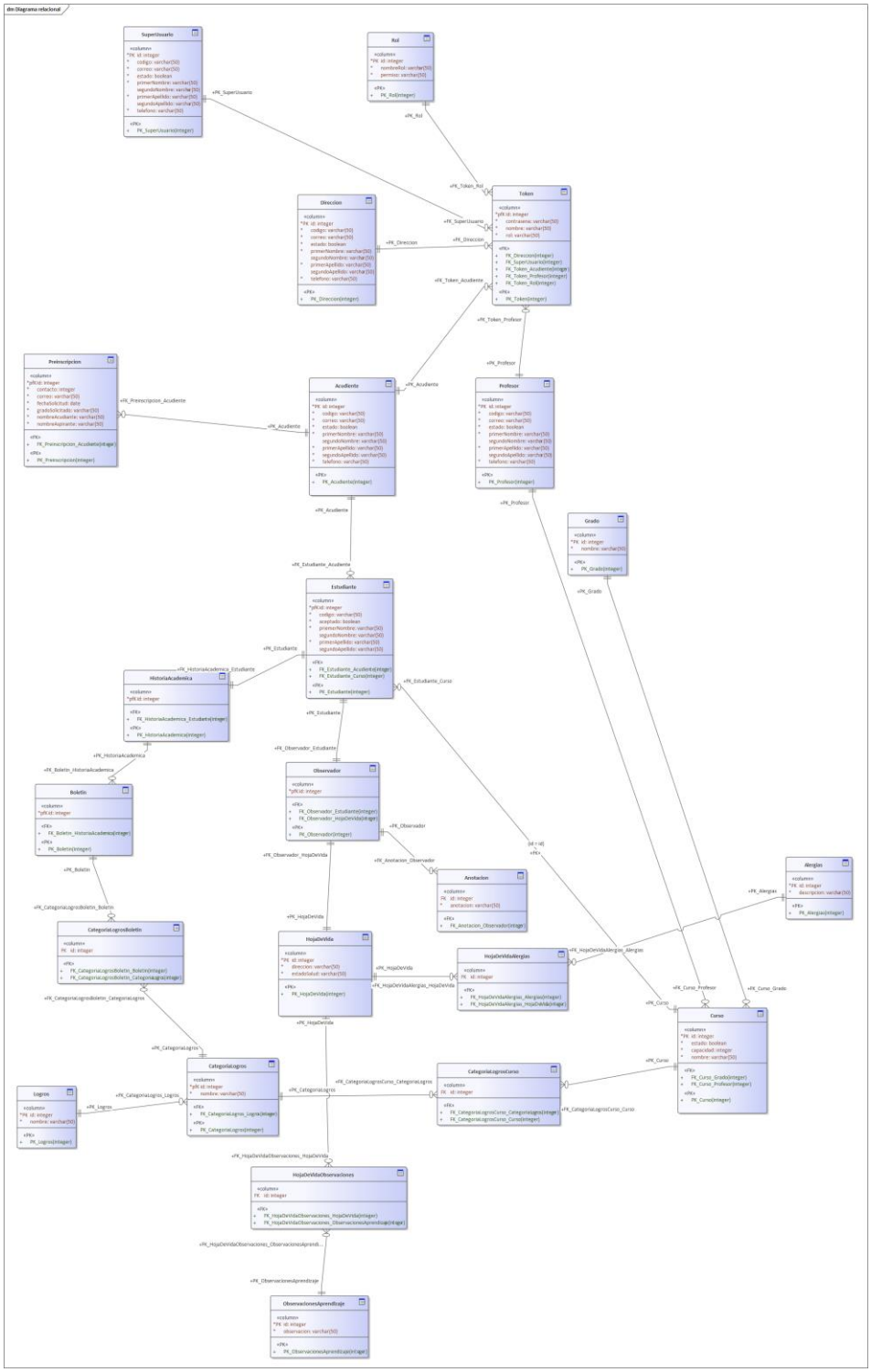
Para el manejo de estructuras de datos que no existen en el modelo relacional como es el caso de las listas o los HashMap fue necesario crear tablas adicionales que guarden la información de dichas estructuras. La información que guardan estas tablas adicionales se puede obtener mediante el uso de llaves foráneas, lo que permite replicar el funcionamiento de las estructuras previamente mencionadas.

En la siguiente figura se ejemplifica la forma cómo se trasladaron las relaciones uno a uno y uno a muchos del modelo de dominio al modelo relacional.



5.2 Diagrama relacional

En esta sección se presenta el diagrama relacional completo de este prototipo, donde se detallan las entidades, atributos, llaves foráneas, llaves primarias, relaciones y restricciones

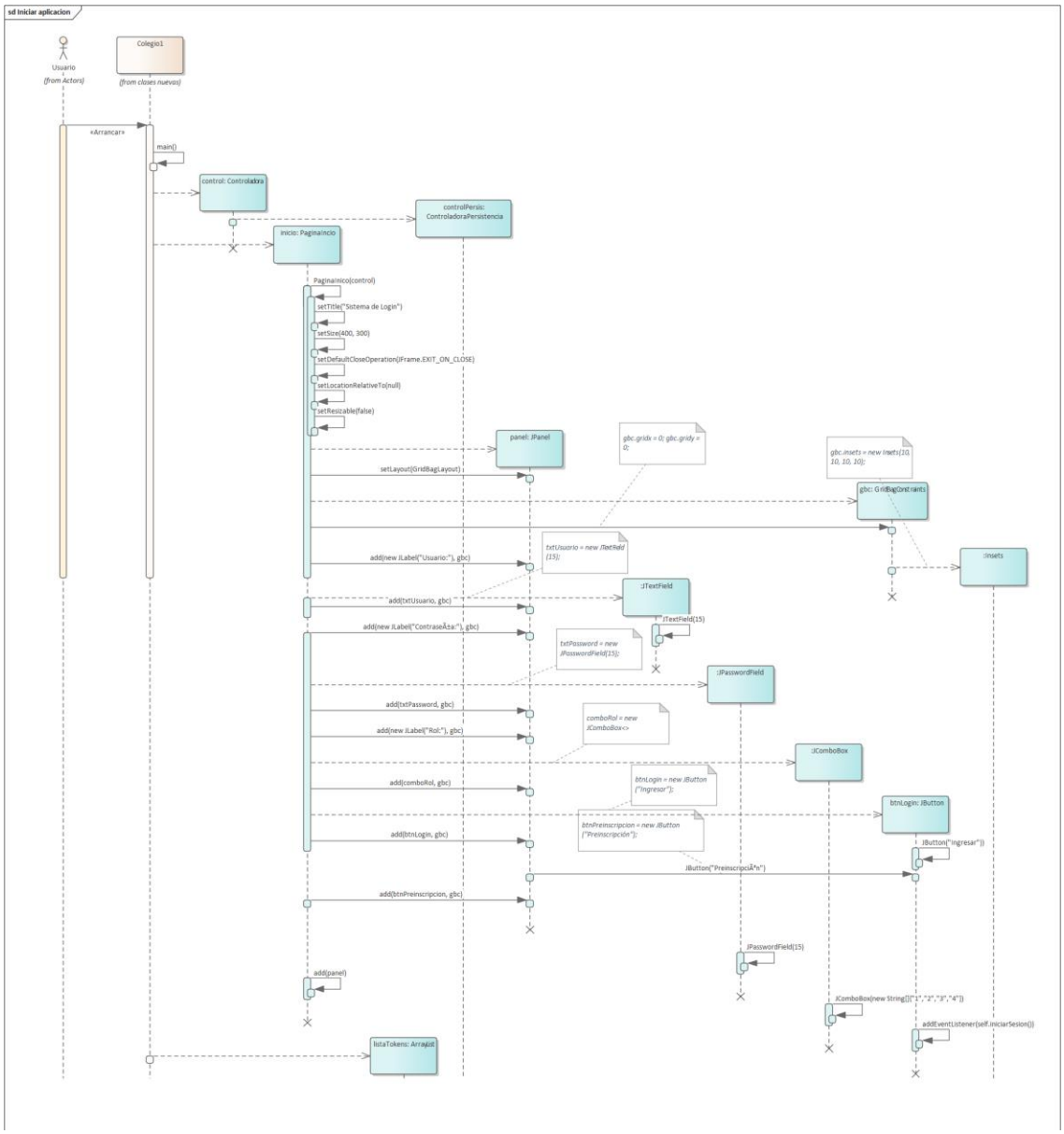

dm Diagrama relacional
SuperUsuario
Rol
Token
Direccion
Preinscripcion
Acudiente
Profesor
Grado
Estudiante
HistoriaAcademica
Observador
Boletin
Anotacion
Alergias
CategoriaLogrosBoletin
HojaDeVida
HojaDeVidaAlergias
Curso
CategoriaLogros
CategoriaLogrosCurso
Logros
HojaDeVidaObservaciones
ObservacionesAprendizaje

CAPITULO 3 MODELO DINÁMICO

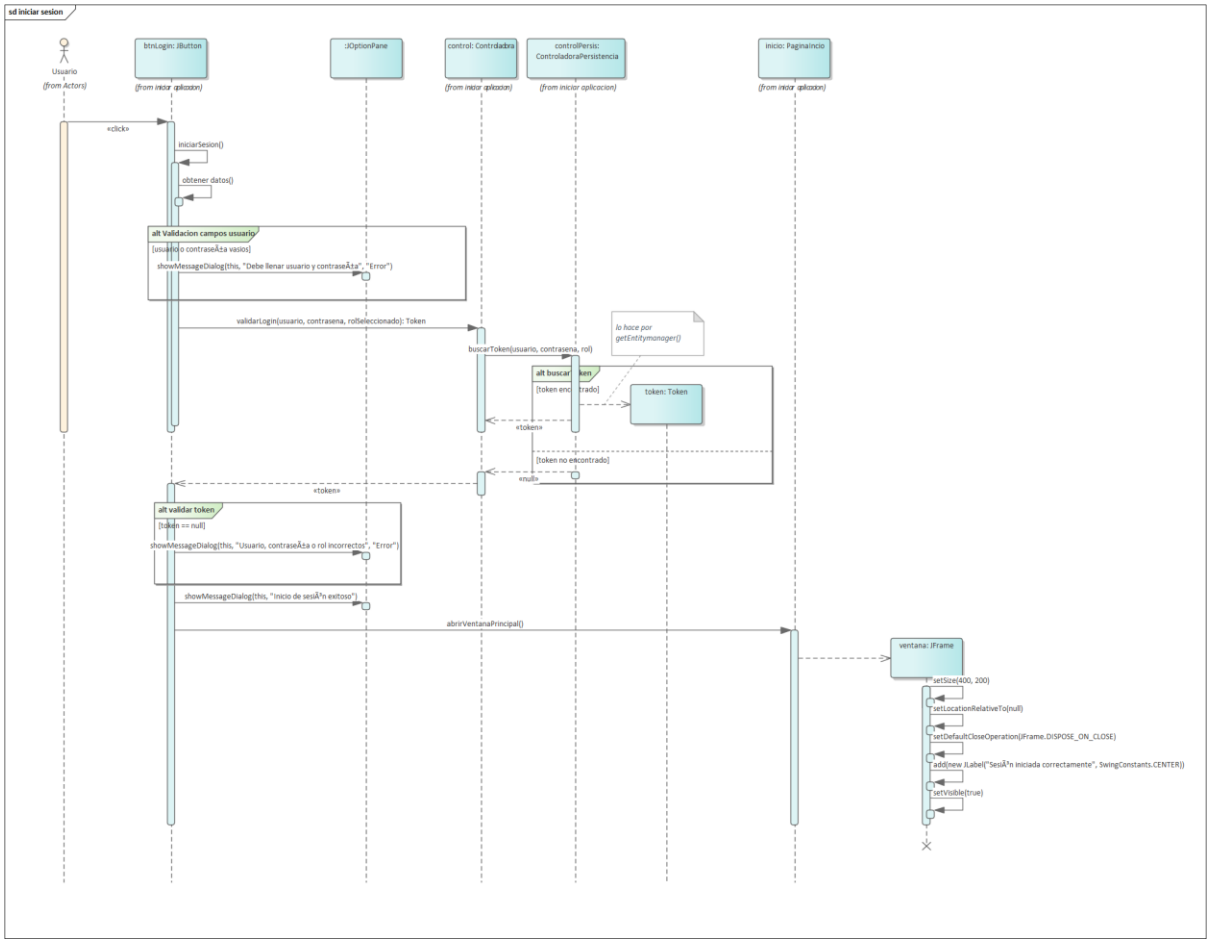
En la presente sección se mostrarán los planos de ingeniería en tiempo de ejecución para algunos casos de uso del sistema propuesto

6. Diagramas de secuencia

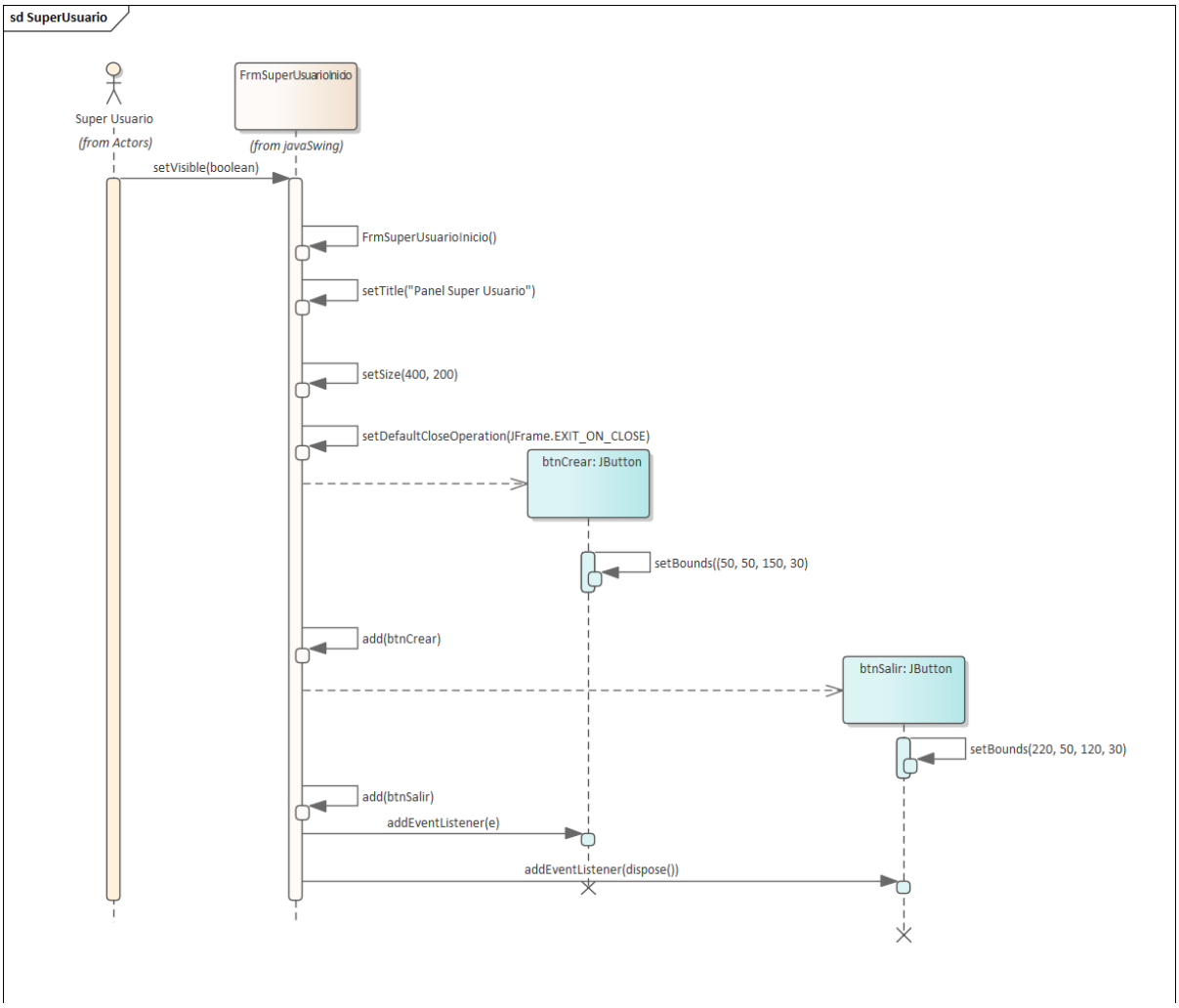
6.1 Iniciar aplicación



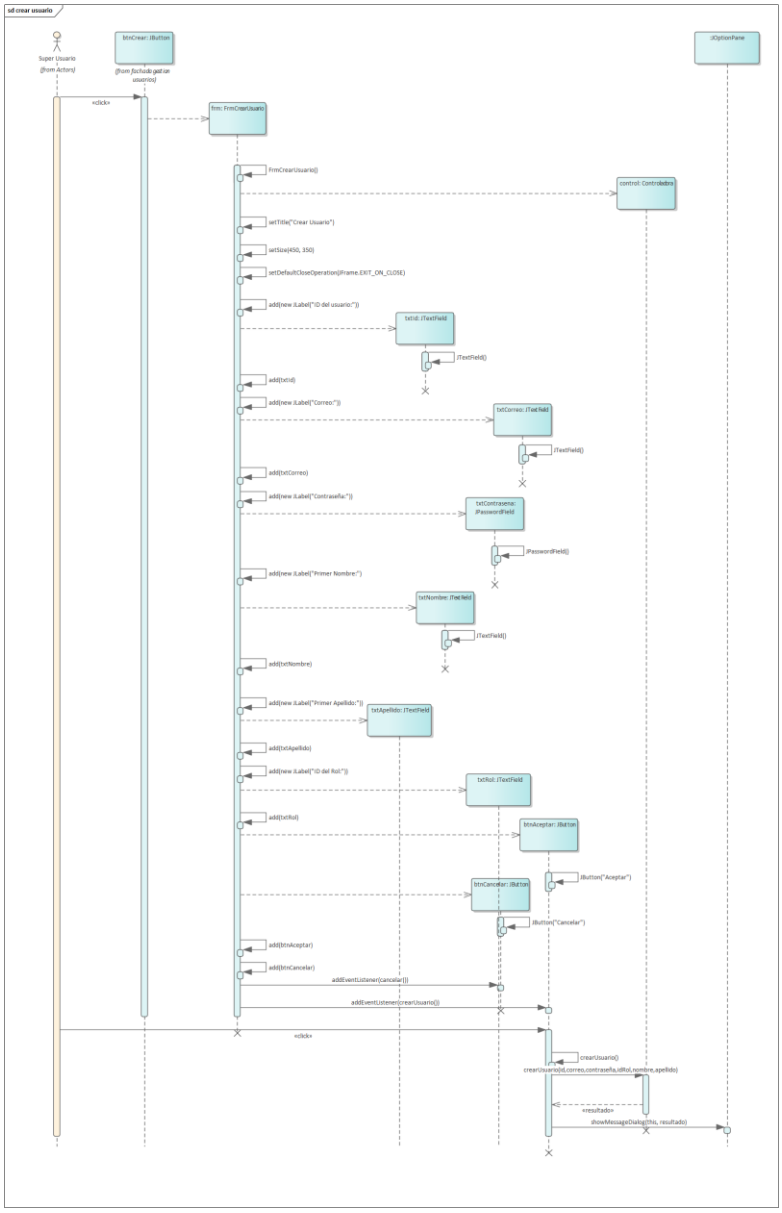
6.2 iniciar sesión



6.2 Fachada gestión de usuario


sd SuperUsuario
Super Usuario
(from Actors)
FrmSuperUsuarioInicio
(from javaSwing)
setVisible(boolean)
FrmSuperUsuarioInicio()
setTitle("Panel Super Usuario")
setSize(400, 200)
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)
btnCrear: JButton
setBounds((50, 50, 150, 30)
add(btnCrear)
btnSalir: JButton
setBounds(220, 50, 120, 30)
add(btnSalir)
addEventListener(e)
addEventListener(dispose())

6.4 Crear usuario



1. Conclusión

El desarrollo del Gestor Académico permitió construir una solución estructurada y coherente para las necesidades operativas de la institución educativa. A través del análisis detallado de roles, procesos y flujos de información, se definió un conjunto de requerimientos completos y trazables que guían el comportamiento del sistema. La

incorporación de modelos funcionales, estructurales y dinámicos permitió visualizar con precisión tanto la interacción de los usuarios como la arquitectura interna que soportará el funcionamiento del software.

La documentación generada establece los cimientos para la etapa de implementación, asegurando que cada vista, módulo y entidad del sistema responda directamente a un requerimiento previamente identificado. El uso de JPA para la persistencia y la aplicación del patrón MVC garantizan una separación clara de responsabilidades, mayor mantenibilidad y una base sólida para futuras mejoras o ampliaciones del sistema.

En conclusión, el proyecto consolida un diseño robusto y orientado a las buenas prácticas de ingeniería de software, ofreciendo una solución que optimiza la gestión académica, facilita el trabajo de los distintos actores de la institución y soporta la evolución del sistema a largo plazo.

2. Bibliografía

Booch, G., Rumbaugh, J., & Jacobson, I. (2005). *The Unified Modeling Language User Guide* (2nd ed.). Addison-Wesley.

Object Management Group (OMG). (2017). *OMG Unified Modeling Language (UML) Specification, Version 2.5.1*.