

**COLEGIO ESPAÑOL PADRE ARRUE
BACHILLERATO TÉCNICO VOCACIONAL**



**SISTEMA INFORMÁTICO PARA LA VIGILANCIA
ESTUDIANTIL Y LOGÍSTICA DE ATENCIÓN**

PRESENTADO POR:

**LUIS GUILLERMO LÓPEZ UMAÑA
PABLO JOSÉ PALACIOS HERNÁNDEZ
HÉCTOR RAÚL SORTO PARADA
DANIEL ALEJANDRO VÁSQUEZ ORELLANA**

PARA OPTAR AL TÍTULO DE:

**BACHILLERATO TÉCNICO VOCACIONAL EN
DESARROLLO DE SOFTWARE**

**CALLE PADRE SALAZAR SIMPSON, SOYAPANGO, OCTUBRE
DE 2024**

COLEGIO ESPAÑOL PADRE ARRUE

DIRECTOR:

SANTIAGO NOGALES

INGENIERO

LUIS ALONSO RIVAS LEIVA

INGENIERO

CARLOS DANIEL CASTILLO SOSA

DOCTORA

ELENA GUADALUPE RECINOS

LICENCIADA

LILIANA LANDAVERDE

**COLEGIO ESPAÑOL PADRE ARRUPE
BACHILLERATO TÉCNICO VOCACIONAL**

Trabajo de Graduación previo a la opción al Grado de:

**BACHILLERATO TÉCNICO VOCACIONAL EN
DESARROLLO DE SOFTWARE**

Título:

**SISTEMA INFORMÁTICO PARA LA VIGILANCIA
ESTUDIANTIL EN LA ATENCIÓN**

Presentado por:

**LUIS GUILLERMO LÓPEZ UMAÑA
PABLO JOSÉ PALACIOS HERNÁNDEZ
HÉCTOR RAÚL SORTO PARADA
DANIEL ALEJANDRO VÁSQUEZ ORELLANA**

Trabajo de Graduación Aprobado por:

Docente Asesor:

CARLOS DANIEL CASTILLO SOSA

SAN SALVADOR, OCTUBRE DE 2024

Agradecimientos:

Quiero agradecer en primer lugar a mi familia por su apoyo incondicional durante todo este proceso. La compresión que me han tenido y el aliento que me han dado han sido fundamentales para alcanzar las metas que me he propuesto.

A mis amigos les doy las gracias por su paciencia y por brindarme momentos de calma y risas y que me han ayudado a mantener el equilibrio durante esta etapa. especialmente a mis amigos Fernando Bandek, Jade Caroline y Alejandro Arriaga, en los que me apoyaron en los buenos y malos momentos y siempre estuvieron ahí.

Expreso mi especial reconocimiento a mi profesor Daniel Sosa, cuya guía y asesoramiento han sido invaluables para el desarrollo de este proyecto. Su conocimiento y experiencia han enriquecido significativamente este trabajo.

Al Colegio Español Padre Arrupe, agradezco por proporcionar el ambiente académico y los recursos necesarios para llevar a cabo este proyecto. La información recibida por la licenciada Liliana Landaverde y la maestra Fátima han sido la base sobre la cual se ha construido este gran proyecto.

A mis compañeros de estudio, con quienes he compartido largas jornadas de trabajo, les agradezco por el intercambio de ideas y el apoyo mútuo, elementos que han contribuido enormemente a mi crecimiento académico y personal.

Finalmente, extiendo mi gratitud a todas aquellas personas que, de una u otra forma, han aportado a la culminación exitosa de este trabajo de investigación.

A todos, mis más sinceros agradecimientos.

Pablo José Palacios Hernández

Agradecimientos:

A mi madre, Rosario del Carmen Umaña, por ser mi pilar incondicional y darme su apoyo en cada paso de mi vida académica y personal. Tu amor, esfuerzo y sacrificio me han dado la fuerza para seguir adelante.

A mi padre, Guillermo Antonio López, por tu sabiduría, tus consejos y el ejemplo de perseverancia que siempre has sido para mí. Gracias por inculcarme el valor del esfuerzo y la dedicación en todo lo que hago.

A mi hermana, Monica Vanessa López, por tu compañía, comprensión y palabras de aliento en los momentos más difíciles. Tu apoyo ha sido esencial para mantenerme enfocado en mis objetivos.

A mi profesor de bachillerato en desarrollo de software, Daniel Sosa, por ser una fuente de inspiración y motivación en mi formación académica. Gracias por tus enseñanzas, por tu guía y por haber confiado en mis capacidades desde el inicio de este camino.

Finalmente, quiero agradecer a la Directiva del Colegio Español Padre Arrupe por brindarme la oportunidad de crecer en un entorno académico de excelencia, y por contribuir a mi desarrollo personal y profesional a lo largo de estos años.

Luis Guillermo López Umaña

Agradecimientos:

Me agradezco a mí mismo en primer lugar, por tener la determinación de nunca rendirme y continuar adelante a pesar de todas las dificultades que se me presentaron, por haber tenido la fortaleza, mentalidad y conocimiento para poder trabajar en este proyecto.

A mi maestro, Daniel Sosa por darme ánimos cuando lo necesitaba, apoyarme y solventar mis dudas y aconsejarme sobre cómo hacer las cosas de la manera correcta para alcanzar mi máximo potencial, a quién le deseo lo mejor de aquí en adelante en su vida y a su familia.

A mis compañeros de trabajo, que cada uno de ellos perseveró y consiguió sus metas y propósitos a lo largo del trabajo y logramos entregar este proyecto de la mejor forma posible.

A mi Mamá que siempre me está apoyando en cada paso de mi vida y busca lo mejor para mí, que he logrado conseguir todo lo que me he propuesto gracias al apoyo que ella me da cada día y ha dado a lo largo de mi vida.

A la lic. Lily y a la maestra Fátima que nos brindaron todo su apoyo para alcanzar nuestro objetivo y la forma sobre cómo estructurar y optimizar este proyecto.

Y especialmente a mi amiga Mónica, quien me ha acompañado durante todo este año, siendo mi mayor motivación para dar lo mejor de mí, quién me acompañó cuando más lo necesitaba y creía en mí cuando yo no podía creer en mí mismo, me acompañó en mis etapas más duras de mi vida, y seguirá estando de aquí en adelante, le deseo lo mejor siempre y que jamás le falte las cosas buenas en la vida, quiero estar con ella por siempre, así como ella ha estado aquí para mí. Gracias.

Héctor Raúl Sorto Parada

Agradecimientos:

Agradezco a Dios todopoderoso por haberme dado sabiduría y le doy las gracias por haberme guiado en este largo camino que he recorrido para poder estar en donde estoy, por haberme cubierto con su manto sagrado para poder salir adelante a pesar de las situaciones que se me presentaban, por ser mi fortaleza para poder mejorar cada dia mas y bendecirme en la vida.

A mi familia, por ser las personas que más amo y aprecio, por ser los que me dieron todo lo necesario para salir adelante, que estuvieron apoyándome durante esta larga etapa, por los consejos que me brindaban, por sus deseos de poder superarme, me hacen ser lo que soy ahora.

A mi profesor, Carlos Daniel Castillo Sosa, por ser la persona que me motivaba a siempre seguir intentándolo, por hacerme salir adelante a pesar de las adversidades que tenía, por ser un buen maestro y mentor para mí y enseñarme todo lo que sé, gracias por todo y estoy infinitamente agradecido con el.

Al Colegio por haberme dado durante la mayor parte de mi vida muchas enseñanzas y aprendizaje, el conocer personas de mucho valor, por haberme inculcado valores, mostrarme sobre la armonía y respeto y sobre todo, por hacerme lo que soy ahora.

Agradezco a mis verdaderos amigos, con los cuales compartí muchas vivencias y que me mostraron que con esfuerzo todo puede ser posible.

Daniel Alejandro Vásquez Orellana

Contenido

Introducción.....	10
Objetivos.....	11
Objetivo General.....	11
Objetivos Específicos.....	11
Capítulo I Anteproyecto.....	13
1.1 Antecedentes.....	13
1.2 Entrevista.....	14
1.3 Situación Problemática.....	15
1.4 Lluvia de ideas.....	15
1.5 Diagramas de causa y efecto.....	16
1.6 Matriz FODA.....	16
1.7 Formulación de problema.....	17
1.8 Importancia.....	17
1.9 Alcances.....	18
1.10 Limitaciones.....	19
1.11 Justificación.....	19
1.12 Resultados esperados.....	20
1.13 Descripción del sistema.....	20
1.13.1 Enfoque de sistema.....	21
1.13.2 Descripción de los componentes del enfoque de sistemas.....	21
1.14 Metodología para desarrollar el proyecto.....	23
1.15 Planificación del Proyecto.....	23
Capítulo II Análisis y Diseño.....	26
2.1 Análisis de la situación actual.....	26
2.2 Diagramas y descripciones de proceso.....	27
2.3 Determinación de requerimientos.....	34
2.3.1 Requerimientos funcionales y no funcionales.....	34
2.3.2 Requerimientos no funcionales.....	38
2.3.3 Requerimientos de software.....	49
2.3.4 Requerimiento de recursos humano.....	50

2.4 Modelo de Dominio.....	50
2.5 Casos de uso del sistema.....	51
2.5.1 Identificación de actores.....	51
2.5.2 Descripción de casos de uso.....	52
2.5.3 Casos de Uso.....	53
2.5.4 Diagramas de casos uso para aplicación móvil y web.....	69
2.6 Historias de usuario.....	73
2.7 Estándares de diseño.....	76
2.9 Diseño de la Base de Datos.....	78
2.9.1 Estructura de tablas.....	79
Capítulo III: Construcción e implementación del sistema.....	85
3.1 Detalles de sistemas.....	85
3.1.1 Códigos explicativos.....	95
3.2 Plan de Implementación.....	128
3.2.1 Método de Implementación.....	128
3.2.2 Configuración e Instalación.....	129
3.2.3 Implementación de un servidor.....	130
3.3 Requisitos mínimos.....	132
Aplicación de escritorio.....	132
Aplicación de celular.....	132
3.4 Conclusiones.....	133
3.5 Recomendaciones.....	134
3.6 Bibliografías.....	135

Introducción

La gestión de sistemas de registro y control es crucial para asegurar una atención médica eficiente y coordinada en entornos educativos, facilitando la organización de datos de salud y mejorando la comunicación entre el personal y los familiares. El sistema informático para la vigilancia estudiantil y logística de atención ha sido desarrollado para optimizar la atención médica de los estudiantes en el Colegio Español Padre Arrupe, delimitándolo específicamente para el Tercer Año Vocacional en Desarrollo de Software, debido a la gran cantidad de datos de estudiantes que hay que manejar. Este sistema permite a administradores, auxiliares de planta y responsables familiares realizar un seguimiento de las intervenciones médicas, la administración de medicamentos y el registro de incidentes de salud, facilitando la toma de decisiones ante emergencias.

El desarrollo del sistema se divide en tres capítulos:

Capítulo 1: Aborda los elementos de anteproyecto, como lo son antecedentes, la formulación del problema, los alcances, limitaciones y los resultados esperados.

Capítulo 2: Analiza la situación actual de atención de incidentes en el colegio, los casos de uso y estándares de diseño.

Capítulo 3: Este capítulo detalla descripciones de los tres sistemas implementados con imágenes y fragmentos de código con comentarios para facilitar la comprensión. También se abordará la documentación necesaria para el uso de estos mismos.

Objetivos

Objetivo General

Optimizar la gestión de la atención médica de los estudiantes en el Colegio Español Padre Arrupe mediante la implementación de un sistema integral que centralice, organice y facilite el acceso a la información médica. Este sistema busca mejorar la eficiencia de los procesos administrativos y médicos, asegurando que todos los involucrados en el bienestar de los estudiantes puedan actuar de manera coordinada y efectiva.

Objetivos Específicos

Facilitar la gestión de la información médica para poder permitir el registro y acceso seguro a los datos de salud de los estudiantes.

Agilizar el proceso de atención médica con un flujo de trabajo que reduzca tiempos de trabajo y mejore la respuesta ante emergencias.

Proporcionar un seguimiento detallado y registrar todas las atenciones médicas y medicamentos administrados de manera segura a los estudiantes.

Capítulo I

Capítulo I Anteproyecto

1.1 Antecedentes

El Colegio Español Padre Arrupe, inaugurado en el año 1998, gracias al esfuerzo y empeño del Padre Juan Ricardo Salazar-Simpson. Amante a la educación y persona que tenía el objetivo formar alumnos íntegros que alcancen una desarrollo personal y comunitario auténtico.

El Colegio se presenta como una institución de convivencia armónica, de valores y respeto para todos los que en él se integran.

En sus inicios, el Colegio comenzó con servicios para los niveles iniciales de educación. Posteriormente, fue ampliando su oferta académica, en el año 2000 se introdujo el Bachillerato Español, en virtud del convenio suscrito con el Ministerio de Educación, Cultura y Deportes de España y recientemente, implementando en el año 2022, el Bachillerato Técnico Vocacional en Desarrollo de Software, con el propósito de brindar una enseñanza de calidad, enfocada en proporcionar a los estudiantes conocimientos avanzados en el área tecnológica, preparando a futuros profesionales con las habilidades necesarias para afrontar los retos del mundo digital. Evaluando las necesidades en cuanto a la seguridad y bienestar de los estudiantes, se decidió implementar un sistema de atención a incidentes relacionados con la salud de los alumnos, que permitiera gestionar situaciones ante algún incidente o enfermedad de manera rápida y eficaz.

Este nuevo sistema informático para la vigilancia estudiantil y logística de atención permite llevar un registro detallado de los incidentes de salud que puedan afectar a los alumnos. A través de esta plataforma, los responsables del colegio pueden responder de manera inmediata a cualquier situación que ponga en riesgo el bienestar de los estudiantes, coordinando de manera eficiente la atención y los cuidados necesarios.

1.2 Entrevista

Se realizó una entrevista a la licenciada Liliana Landaverde y la maestra Fátima, se destacó que la información que manejan es delicada. Es por eso que es crucial que la institución conozca las condiciones médicas de los estudiantes para garantizar el cuidado adecuado, ya que algunos medicamentos pueden ser perjudiciales para ciertos alumnos, es por eso que se nos detallo los siguientes apartados que se pueden abarcar

Registrar incidente

Se pregunta al estudiante si ha ingerido alimentos, si es alérgico a medicamentos, y sí ha tomado pastillas anteriormente (sea el caso para las enfermedades tales como dolor de cabeza, dolor de cuerpo, náusea, dolor de estómago). Si sufre de otra condición, se especifica.

Perfil médico

Aquí se proporcionan los datos de los responsables y condiciones médicas que el estudiante sufre. Para los responsables, se obtiene el nombre, número de teléfono y correo electrónico.

Para el estudiante se proporciona los datos como: Sí tiene condición crónica, si toma medicamentos permanentes (en caso de no continuar el medicamento, se especifica de igual forma), a qué es alérgico, si es alérgico a medicamentos. El responsable debe de aceptar que el colegio sea capaz de brindarle los medicamentos que tienen, los cuáles se especifican a continuación.

El colegio está autorizado únicamente de proporcionar las siguientes medicinas: Acetaminofén, Clorfenamina, Ibuprofeno, Anaflat, Neobol, Volfenac y Betamed. La licenciada Liliana Landaverde y la maestra Fátima y en ocasiones la Jefa de Estudios de Tercer Ciclo y Bachillerato Elena Guadalupe Recinos , autorizan brindar los medicamentos, si el estudiante no ha ingerido alimentos o, si ya tomó medicamentos, no se le brindan más.

1.3 Situación Problemática

El Colegio Español Padre Arrupe se enfrenta al reto de optimizar la gestión del bienestar y la salud de sus estudiantes. A medida que la comunidad educativa ha crecido, también lo han hecho las necesidades de atención médica y la capacidad de respuesta ante incidentes de salud. Situaciones como malestares repentinos, alergias, accidentes menores o condiciones médicas crónicas requieren una intervención rápida y eficaz. Sin embargo, la implementación de un sistema más organizado para registrar y coordinar estas situaciones ofrecería una mayor eficiencia en la atención de cada caso.

1.4 Lluvia de ideas

Sistema Digital de Registro Médico:

No se ha optimizado una plataforma digital donde se registren incidentes médicos.

Sistema de Identificación de Condiciones Médicas Crónicas:

No se ha facilitado la búsqueda sencilla de un identificador principal (Carné) de los alumnos en el sistema, lo que puede llegar a ser más complejo, esto puede ayudarnos a tener más rapidez para las búsquedas.

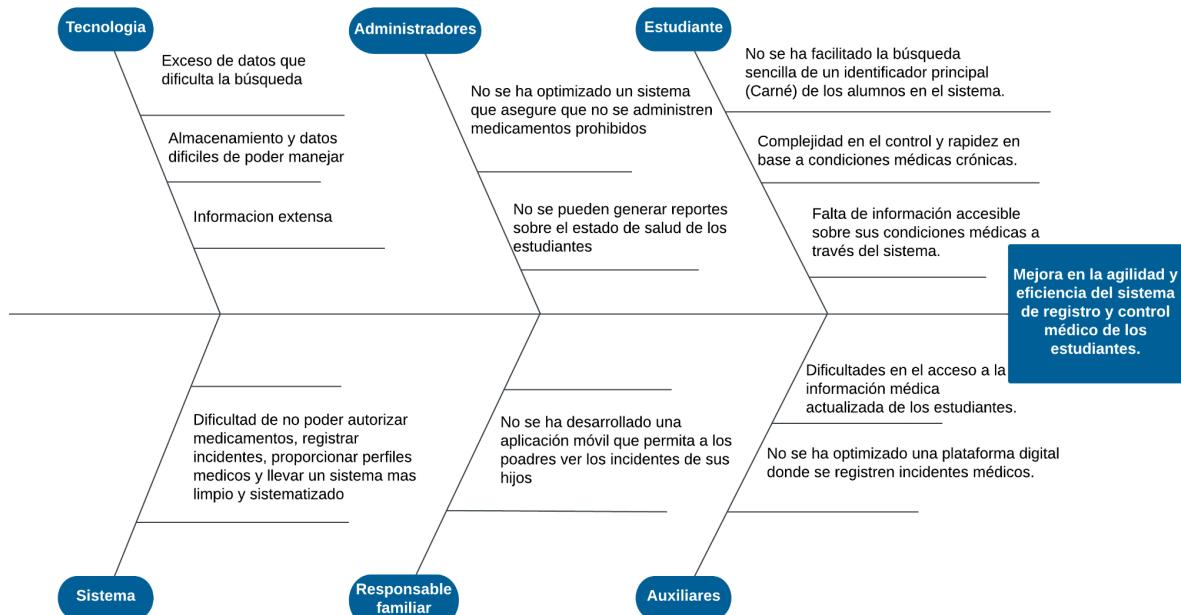
Aplicación Móvil para Responsables Familiares:

No se ha desarrollado una aplicación móvil que permita a los padres consultar el estado de salud de sus hijos y notificar la necesidad de retirarlos del colegio por causa mayor.

Control de Medicamentos:

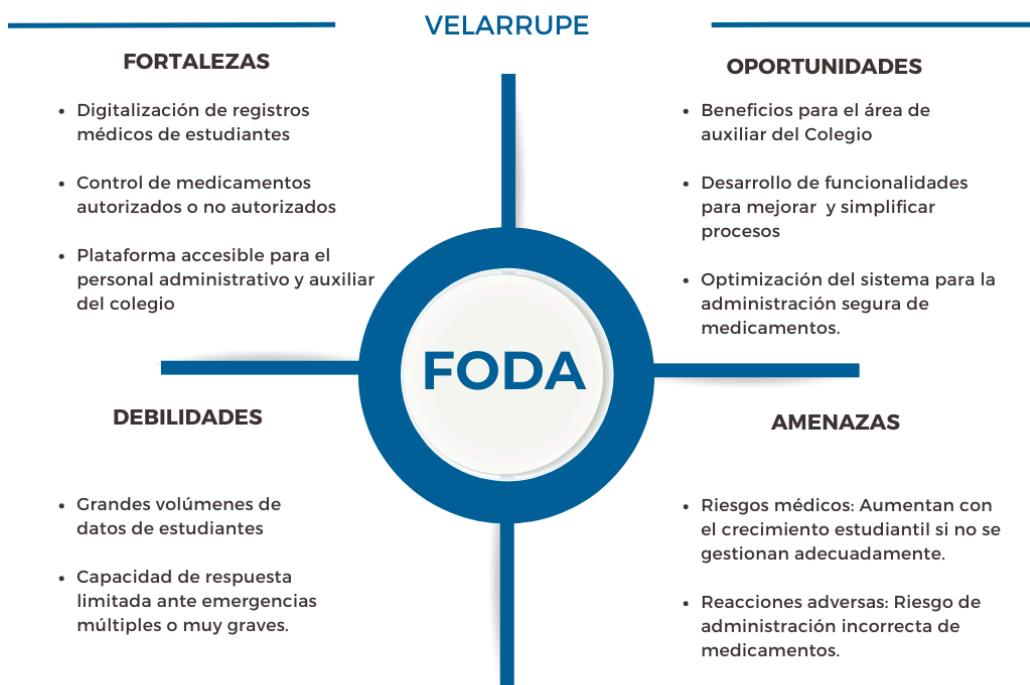
No se ha optimizado un sistema que asegure que no se administren medicamentos prohibidos o que puedan causar alergias o efectos negativos a los estudiantes.

1.5 Diagramas de causa y efecto



1.6 Matriz FODA.

Desarrollar un Sistema Informático para el Registro y Control de Reportes Auxiliares para gestionar eficientemente la atención médica de estudiantes en el Colegio Español Padre Arrupe.



1.7 Formulación de problema

Para la formulación del problema, se toma en cuenta las técnicas utilizadas anteriormente, comenzando por la lluvia de ideas que involucra a auxiliares, responsable familiar y personal administrativo del colegio. Estas partes necesarias expresan sus necesidades, lo que ayuda a identificar la problemática y también proporciona información valiosa para el análisis FODA. Este análisis permite evaluar los factores internos y externos que afectan la gestión de la salud de los estudiantes en el colegio.

Con base en las técnicas utilizadas para el análisis del problema, se define la siguiente problemática:

¿En qué medida el desarrollo del Sistema Estudiantil para la Atención ayudará a mejorar la atención médica, el seguimiento de las condiciones de salud y la administración de medicamentos en el Colegio Español Padre Arrupe, para garantizar que se atiendan de manera oportuna las necesidades de salud de los estudiantes y se minimicen los riesgos asociados a su bienestar?

1.8 Importancia

La importancia del proyecto radica en los beneficios que se obtendrán con la implementación del sistema digital de gestión de incidentes, los cuales contribuirán a contar con un proceso más eficiente para la atención del estado de los estudiantes en el colegio.

El impacto de este sistema se reflejará en la reducción del tiempo que el personal auxiliar y administrativo invierte en el registro de incidentes médicos, la revisión de historiales de salud, y el control de la administración de medicamentos. La digitalización de estos procesos permitirá una respuesta más rápida y efectiva ante las necesidades médicas de los estudiantes.

Los responsables familiares, pueden visualizar los incidentes de sus hijos, esto les permitirá estar más informados sobre el bienestar de estos mismos y facilitará la comunicación con el personal administrativo del colegio.

Además, existirá un mayor control sobre las condiciones de salud de los estudiantes, lo que garantizará que se atiendan adecuadamente sus necesidades médicas y se eviten situaciones críticas.

1.9 Alcances

Soporte al Proceso de Registro de Incidentes:

Permitirá el ingreso de la información ante complicaciones que presenten los estudiantes, incluyendo antecedentes médicos, alergias, condiciones crónicas y cualquier intervención realizada en el colegio.

Soporte al Proceso de Atenciones Médicas:

Facilitará el registro de consultas médicas y administración de medicamentos, con fechas y detalles gestionados por el personal autorizado del colegio.

Soporte al Proceso de Intervenciones Médicas:

Permitirá a los auxiliares registrar de manera detallada todas las intervenciones médicas realizadas a los estudiantes, especificando síntomas, tratamientos y cualquier recomendación adicional.

Soporte al Proceso de Comunicación Familiar:

Incluirá un apartado que permita a los responsables familiares visualizar las atenciones médicas recibidas a sus hijos.

Soporte al Proceso de Seguimiento Médico:

Permitirá realizar un seguimiento continuo de las condiciones de salud de los estudiantes, proporcionando un registro de las atenciones médicas previas y las recomendaciones para un control más eficiente y efectivo por parte de los auxiliares y personal médico.

1.10 Limitaciones

1. No puede abarcar un grupo muy grande de estudiantes por el momento.
2. No es funcional con iOS.

1.11 Justificación

La implementación del sistema de vigilancia estudiantil y logística de atención en el Colegio Español Padre Arrupe es crucial para mejorar la atención médica y el bienestar de los estudiantes. Un sistema automatizado permitirá centralizar y organizar los datos de los estudiantes, pudiendo acelerar y optimizar el tiempo.

Además, este sistema proporcionará una mejor comunicación entre la institución educativa y los responsables familiares, quienes podrán visualizar la situación de sus hijos. Esto no solo refuerza la confianza de los padres en la seguridad y el cuidado que el colegio brinda, sino que también mejora la coordinación ante posibles emergencias o tratamientos continuos.

La digitalización de estos procesos no sólo optimizará la labor de los auxiliares de planta y el personal administrativo, sino que también asegurará el cumplimiento de las normativas adecuadas de la información, protegiendo los datos sensibles de los estudiantes por parte del administrador.

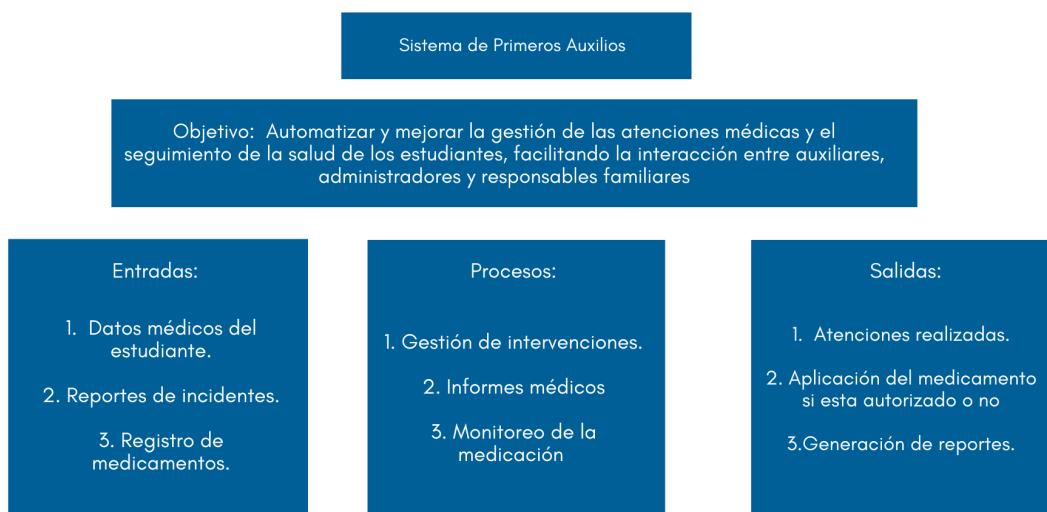
1.12 Resultados esperados

Nivel Administrativo	Salida	Descripción
Operativo	Informe de Atención Médica	Documento que contiene la información sobre la atención médica brindada a un estudiante, detallando el motivo de la consulta y los medicamentos aplicados
Táctico	Reporte de Atenciones Médicas Realizadas	Listado detallado de las atenciones médicas realizadas
	Historial Médico del Estudiante	Reporte que contiene el historial completo de las atenciones médicas de un estudiante, incluyendo las intervenciones previas, tratamientos y medicamentos administrados
Estratégico	Reportes de Atenciones Médicas	Informes en pdf que muestran los incidentes que presentan los estudiantes durante la jornada lectiva

1.13 Descripción del sistema

La descripción del sistema de vigilancia estudiantil y logística de atención se realizará con un enfoque de sistema, como se muestra a continuación.

1.13.1 Enfoque de sistema



1.13.2 Descripción de los componentes del enfoque de sistemas

Entradas:

Datos médicos del estudiante

Información médica relevante del estudiante, como historial médico, alergias, enfermedades crónicas, entre otros.

Reportes de incidentes

Información sobre cualquier incidente de salud que ocurra en el colegio (enfermedades o emergencias médicas).

Registro de medicamentos

Información sobre los medicamentos administrados a los estudiantes.

Procesos:

Gestión de intervenciones

Manejo y seguimiento de las atenciones médicas brindadas a los estudiantes, incluyendo los medicamentos adecuados.

Monitoreo de la medicación

El personal de auxiliar y administración deben asegurar la correcta administración de los medicamentos autorizados según lo que presenta un estudiante.

Generación de reportes

Generación de registros basado en el incidente que presenta el estudiante.

Salidas:

Atenciones realizadas

Registro de todas las intervenciones médicas realizadas a los estudiantes.

Aplicación del medicamento si está autorizado o no

Administración de medicamentos según autorización previa y condición del estudiante.

Generación de reportes

Creación de informes en PDF sobre las fichas.

1.14 Metodología para desarrollar el proyecto

Para el desarrollo del sistema se utilizará la metodología en Cascada, al ser un marco de trabajo para desarrollo ágil de software.

El enfoque en Cascada organiza el desarrollo del sistema de manera iterativa y ágil, además de que ordena rigurosamente las etapas del proceso para el desarrollo de software, de tal forma que el inicio de cada etapa debe esperar a la finalización de la etapa anterior.

1.15 Planificación del Proyecto

A continuación, se presenta la cuantificaciones del costeo a utilizar durante el desarrollo del proyecto.

Recursos Humanos

En esta sección se detalla el costo asociado al equipo de trabajo asignado al proyecto

Recurso	Costo(USD)	Descripción	Cantidad	Horas Trabajadas
Recursos Humanos	50	Trabajo de los sistemas	4 personas	720

Recursos de Computación

Esta sección describe el costo de equipos utilizados para el desarrollo del proyecto provenientes de la Fundación Padre Arrupe

Recurso	Costo(USD)	Descripción	Modelo	Cantidad
Recursos de Computación	2,000	Adquisición de equipos	Dell Latitude 3420	4

Licencias de Software

En esta sección se detalla el gasto destinado a la adquisición de licencias de software necesarias para el proyecto

Recurso	Costo(USD)	Descripción	Cantidad
Licencias de Software	1,559.6	Windows Empresarial	4

Costo de Software

En esta sección se presenta la estimación del costo asociado a las aplicaciones desarrolladas por parte del equipo de trabajo en el marco del proyecto.

Recurso	Costo(USD)	Descripción	Cantidad
Costo de Software	36 mil	Costo total de aplicaciones	3

Capítulo II

Capítulo II Análisis y Diseño

2.1 Análisis de la situación actual

El Colegio Español Padre Arrupe cuenta con un amplio sistema educativo que abarca diferentes niveles académicos, desde los niveles de cuarto, quinto y sexto año de vida, Educación básica de 1° a 9° grado diferenciando en los niveles educativos de primero, segundo y tercer ciclo y bachillerato en las especialidades de Bachillerato General, Bachillerato Técnico Vocacional en Contaduría , Técnico Vocacional Electrónico y Técnico Vocacional en Desarrollo de Software.

El proceso de atención médica comienza con las intervenciones en casos de algún síntoma o situación que el estudiante esté presentando durante el horario lectivo dentro de la institución, se generan reportes, se tiene administración de medicamentos, y registro de incidentes de salud. Los auxiliares registran y gestionan los datos de los estudiantes, incluyendo reportes de incidentes, administración de medicamentos, gestión de fichas médicas, vinculación del responsable familiar con su hijo.

Este nuevo sistema permite un control y seguimiento adecuado de los estudiantes, ofreciendo una solución integral para la gestión de la salud escolar, pudiendo optimizar la coordinación entre auxiliares, administradores, y responsables familiares.

2.2 Diagramas y descripciones de proceso

Descripción de proceso

Para la descripción de los procesos se utiliza el siguiente formato

Proceso: nombre de proceso		
Actores principales: se listan los actores involucrados en el proceso		
No.	Actividad	Responsable
Paso N	Descripción de los pasos para realizar el proceso	Actor que realiza la actividad

Diagrama de Procesos

Para representar los procesos se utilizan los siguientes elementos que componen los diagramas de procesos.

Simbolo	Descripcion
	Inicio: se utiliza para indicar el inicio del diagrama, de este simbolo solo puede salir una linea.
	Actividad: Indica una acción general que debe realizarse en el flujo de trabajo.
	Decisión: Indica la comparación de 2 datos y dependiendo del resultado lógico (verdadero o falso) se toma la decisión de tomar un camino o el otro en el diagrama.
	Flujo: indica el seguimiento lógico del diagrama, también indica el sentido de ejecución de las operaciones.
	Participante: representa un actor del flujo dentro del diagrama, es decir un actor o entidad específica del negocio.
	Descripción: Se detalla las acciones que se tienen sobre el.
	Acción: Representa las actividades específicas que un participante realiza para avanzar en el flujo de trabajo.
	Fin: se utiliza para indicar el fin del diagrama, de este simbolo solo debe de entrar una linea.

Se ha decidido tomar las 3 interacciones principales para poder abarcar los siguientes diagramas de proceso

- 1- Diagrama de Estudiante
- 2- Diagrama de Auxiliar
- 3- Diagrama de Responsable Familiar

Diagrama de atención al estudiante

Ilustra el flujo del proceso de atención médica que sigue un estudiante al enfrentar un incidente de salud

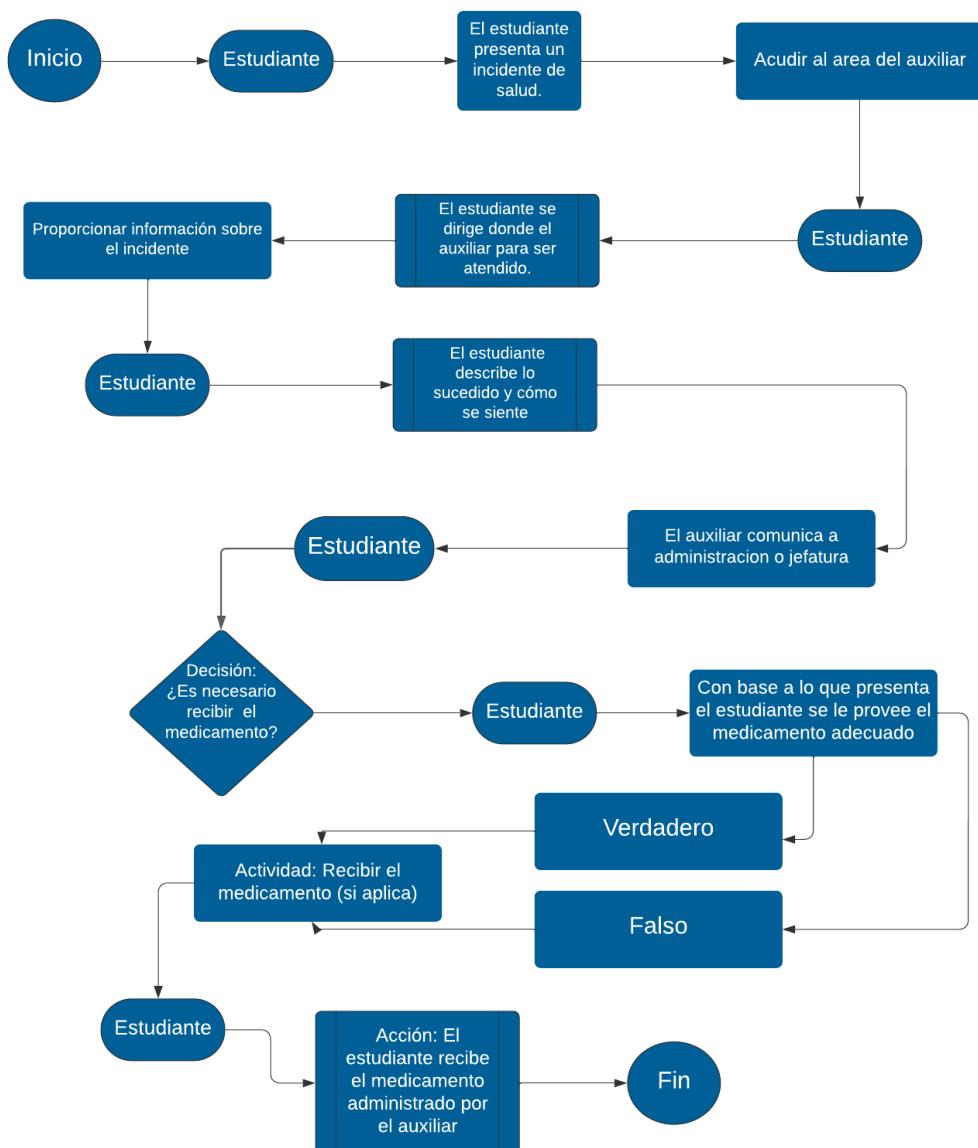


Tabla de gestión de atención al estudiante

La tabla de gestión de atención al estudiante detalla el proceso para atender incidentes de salud, asegurando que el estudiante reciba la atención necesaria.

Proceso : Acción del Estudiante		
Actores principales: Estudiante		
No.	Actividad	Responsable
1	El estudiante presenta un incidente de salud.	Auxiliar
2	El estudiante se dirige al área médica.	
3	Proporciona información sobre el incidente y cómo se siente	
4	Recibe el medicamento (si es necesario).	

Diagrama de atención del auxiliar

El diagrama de atención del auxiliar representa visualmente el flujo del proceso de atención que sigue un auxiliar al asistir a un estudiante con un incidente de salud

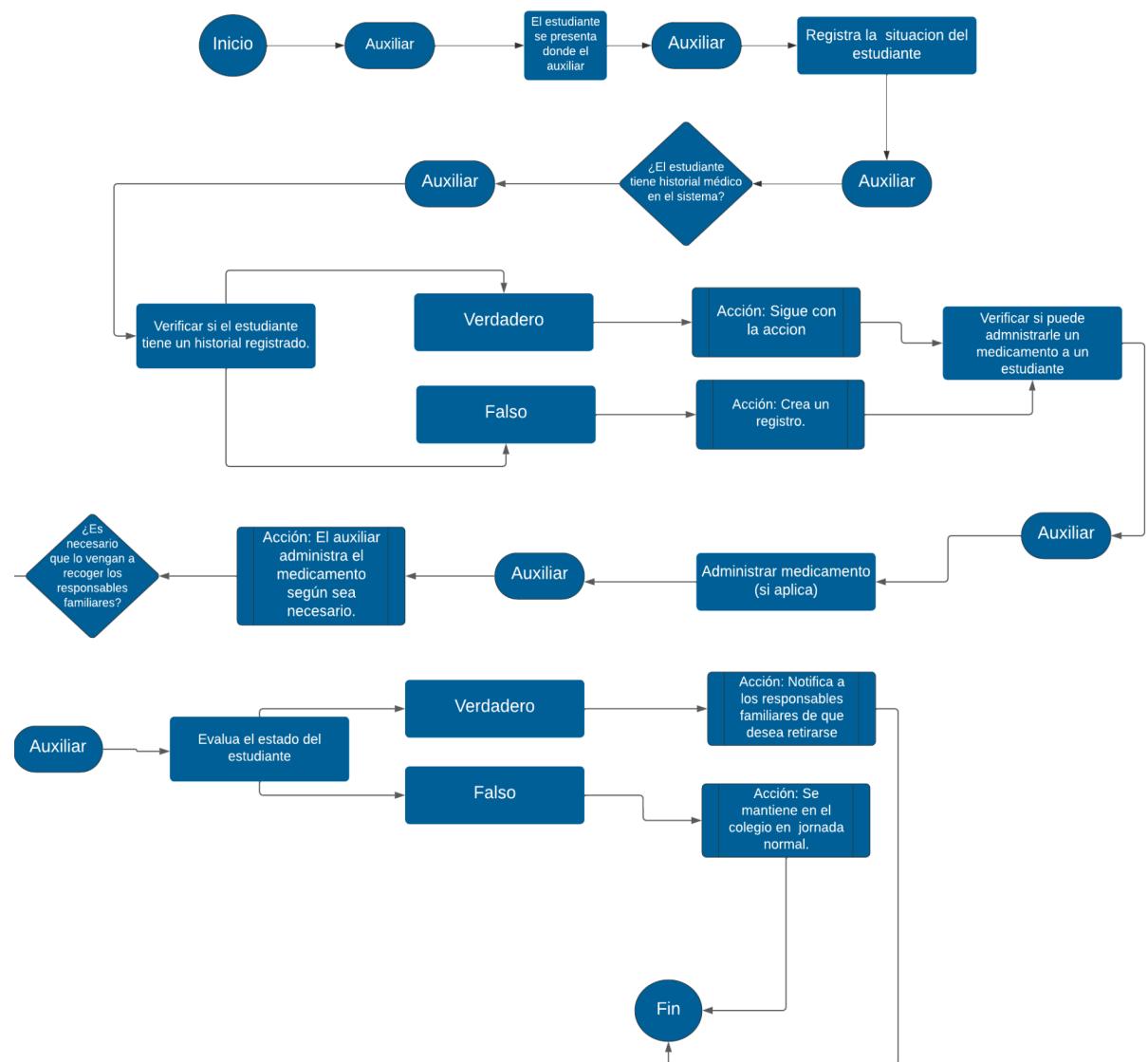


Tabla de atención del auxiliar

La tabla de atención del auxiliar detalla el proceso que sigue un auxiliar al atender a un estudiante que presenta un incidente de salud.

Proceso : Atención del auxiliar		
Actores principales: Auxiliar		
No.	Actividad	Responsable
1	El estudiante se presenta en el área de auxiliar	Auxiliar y Administrador
2	Se le pregunta al estudiante sus síntomas o la situación que presenta	
3	Realiza la evaluación con base a lo que presenta estudiante	
4	Se le comunica al administrador si es necesario administrar medicamento	
5	Administra medicamento, si aplica	

Diagrama de responsable familiar para supervisar al estudiante

El diagrama de responsable familiar para supervisar al estudiante visualiza el proceso que sigue un familiar al supervisar la atención del estudiante.

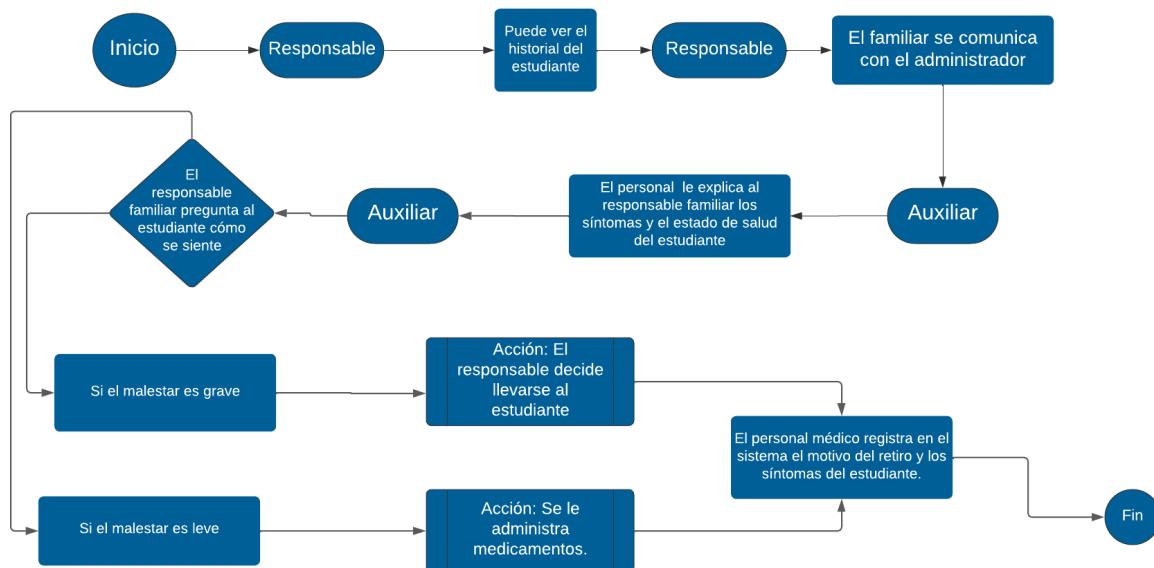


Tabla de Responsable supervisión con estudiante

La tabla de responsable supervisión con estudiante describe el proceso que sigue un familiar al supervisar la atención del estudiante

Proceso : Responsable familiar para supervisar al estudiante		
Actores principales: Responsable familiar		
No.	Actividad	Responsable
1	Ve el historial del estudiante	Administrador
2	Se comunica con el administrador para saber el estado del estudiante	
3	El personal le explica a los responsables los síntomas y el estado de salud del estudiante	
4	Si el malestar es grave, el responsable decide llevarse al estudiante, si es leve, se le administra medicamentos autorizados	

2.3 Determinación de requerimientos

2.3.1 Requerimientos funcionales y no funcionales

Esta tabla muestra los detalles sobre los requisitos informáticos para garantizar su funcionalidad dentro del sistema.

Requerimiento Funcional N° X	
Función	Función a realizar.
Descripción	Descripción de la función a realizar.
Entradas	Entradas necesarias para cumplir con el requerimiento.
Fuente	De donde provienen las entradas requeridas.
Salida	Resultado a ser obtenido de la realización del requerimiento.
Destino	Lugar hacia donde se dirigen las salidas producidas.
Acción	Pasos necesarios para llevar a cabo la función.
Requisitos	Requisitos utilizados para la función.
Precondición	Condiciones que deben cumplirse para satisfacer la función.

1. Aplicación de escritorio

Tabla de requerimiento funcional 1

La tabla de requerimiento funcional 1 detalla la gestión de perfiles médicos, permitiendo a los auxiliares buscar y gestionar información sobre alergias y enfermedades de los estudiantes, utilizando su nombre o carné, con la condición de que estén registrados previamente.

Requerimiento funcional Nº1	
Función	Gestión de Perfiles Médicos.
Descripción	El sistema permitirá a los auxiliares buscar, visualizar y gestionar los perfiles médicos de los estudiantes, incluyendo datos como alergias, enfermedades crónicas, medicamentos crónicos y responsables.
Entradas	Término de búsqueda (nombre o carné del estudiante).
Fuente	Base de datos con base a las tablas estudiantes y fichas médicas y manejando con API local.
Salida	Perfiles médicos filtrados con detalles .
Destino	Auxiliares del sistema.
Acción	Búsqueda de estudiantes por nombre o carne, despliegue de fichas médicas completas.
Requisitos	Conexión con la API del sistema, filtrado eficiente.
Precondición	Los estudiantes deben ser registrados por el administrador previamente en el sistema.

Tabla de requerimiento funcional 2

La tabla de requerimiento funcional 2 describe la gestión de usuarios, que permite a los auxiliares acceder al sistema validando su identidad con un nombre de usuario y contraseña. Si el usuario está registrado, se le permitirá el acceso; de lo contrario, se rechazará.

Requerimiento funcional Nº2	
Función	Gestion de Usuarios (Login)
Descripción	El sistema valida el rol auxiliar, permitiendo el acceso a las funcionalidades correspondientes.
Entradas	Nombre de usuario y contraseña.
Fuente	Servidor (API)
Salida	Autenticación exitosa o rechazo del acceso.
Destino	Usuario del sistema (auxiliares).
Acción	Autenticación y control de acceso.
Requisitos	Integración de la API como intermediario de la tabla usuarios de la base de datos.
Precondición	Usuarios pre-registrados en el sistema.

Tabla de requerimiento funcional 3

La tabla de requerimiento funcional 3 se centra en el registro de incidentes médicos, permitiendo a los auxiliares registrar información sobre los incidentes de salud de los estudiantes, incluyendo detalles como enfermedad, última comida y solicitud de retiro.

Requerimiento funcional Nº3	
Función	Registro de Incidentes médicos
Descripción	El sistema permitirá registrar incidentes médicos de los estudiantes con detalles como enfermedad, última comida y solicitud de retiro del estudiante
Entradas	Nombre de usuario y contraseña
Fuente	Servidor (API)
Salida	Autenticación exitosa o rechazo del acceso
Destino	Usuario del sistema (auxiliares)
Acción	Autenticación y control de acceso
Requisitos	Integración de la API como intermediario de la tabla usuarios de la base de datos
Precondición	Usuarios pre-registrados en el sistema

2.3.2 Requerimientos no funcionales

Tabla de requerimiento no funcional 1

La tabla de requerimiento no funcional 1 aborda la disponibilidad del sistema, que debe estar operativo durante los días de clase y horas laborales. Esto asegura que auxiliares y personal administrativo tengan acceso al sistema en momentos críticos. Se requiere un servidor con alta disponibilidad y un mantenimiento adecuado.

Requerimiento no funcional Nº1	
Función	Disponibilidad
Descripción	El sistema debe de estar disponibles durante los días de clase durante las horas de trabajo
Entradas	N/A
Fuente	N/A
Salida	N/A
Destino	Auxiliares, administrativos
Acción	Garantizar alta disponibilidad del sistema durante los períodos críticos
Requisitos	Alta disponibilidad del servidor (API)
Precondición	Mantenimiento adecuado del servidor y de la base de datos

2. Aplicación móvil

Tabla de requerimiento funcional 1

La tabla de requerimiento funcional 1 detalla la gestión de medicamentos permitidos, dando así la autoridad a los auxiliares de gestionar el control y proceso que se lleve a cabo por los incidentes que puede pasar a los estudiantes, esto les permitirá saber que medicamento tienen autorizados tomar los estudiantes dentro de la institución.

Requerimiento funcional Nº1	
Función	Gestión de Medicamentos Permitidos.
Descripción	El sistema permitirá a los auxiliares llevar a cabo el proceso de darle un medicamento al estudiante, tomando en cuenta cuales son los permitidos por el responsable familiar claro.
Entradas	Permisos de los estudiantes al permitir los medicamentos.
Fuente	Base de datos con base a las tablas estudiantes y fichas médicas y manejando con API local.
Salida	Medicamentos permitidos por estudiante .
Destino	Auxiliares del sistema.
Acción	Observar cuáles medicamentos tienen permitidos tomar el estudiante dentro de la institución.
Requisitos	Conexión con la API del sistema, filtrado eficiente.
Precondición	Los estudiantes deben ser registrados por el administrador previamente en el sistema.

Tabla de requerimiento funcional 2

La tabla de requerimiento funcional 2 detalla la gestión de usuarios en el sistema, esto permite a los administradores gestionar el control para añadir, eliminar, editar un nuevo usuarios el cual tendrá una funcionalidad en el sistema, de esto solo tiene el privilegio el administrador, ya que el decide quien puede entrar al sistema y con qué tipo de rol.

Requerimiento funcional Nº2	
Función	Gestión de Usuarios.
Descripción	El administrador será capaz de controlar y manejar a los usuarios dentro del sistema.
Entradas	Llenar los campos requeridos para poder registrar a un usuario, luego podremos editarlos o eliminarlos dependiendo de los altos cargos.
Fuente	Base de datos con base a las tablas estudiantes y fichas médicas y manejando con API local.
Salida	Datos de los usuarios.
Destino	Auxiliares del sistema.
Acción	Llevar el control de todos los usuarios del sistema, así evitar cualquier problema al momento de realizar un cambio.
Requisitos	Conexión con la API del sistema, filtrado eficiente.
Precondición	Los usuarios deben ser registrados por el administrador previamente en el sistema.

Tabla de requerimiento funcional 3

La tabla de requerimiento funcional 3 detalla la gestión de incidentes reportados para informar a los responsables familiares en tiempo real para llevar un control sobre los reportes que se hacen por parte de la institución y así informar de forma directa al responsable familiar del estudiante lo que ha pasado y como ha pasado.

Requerimiento funcional Nº3	
Función	Gestión de Reportes Incidentes.
Descripción	El usuario responsable familiar podrá ver en su cuenta el nombre del estudiante de quién es responsable y si le sucede algún incidente y que tipo de incidente le sucede, la fecha en la que pasó y si le aceptan el retiro a su estudiante.
Entradas	Visualizar todos los datos del estudiante con respecto a los reportes de incidentes que tiene registrados.
Fuente	Base de datos con base a las tablas estudiantes y fichas médicas y manejando con API local.
Salida	Datos de los reportes de incidentes de los estudiantes.
Destino	Responsable familiar del sistema.
Acción	Informar en tiempo real si le sucede algún incidente a su estudiante (hij@) dentro de la institución.
Requisitos	Conexión con la API del sistema, filtrado eficiente.
Precondición	Tiene que haber estudiantes y usuarios tipo responsables familiares previamente registrados y vinculados en la base de datos.

Tabla de requerimiento funcional 4

La tabla de requerimiento funcional 4 detalla la gestión de perfiles médicos, permitiendo a los auxiliares buscar y gestionar información sobre alergias y enfermedades de los estudiantes, utilizando su nombre o carné, con la condición de que estén registrados previamente.

Requerimiento funcional Nº4	
Función	Gestión de Perfiles Médicos.
Descripción	El sistema permitirá a los auxiliares buscar, visualizar y gestionar los perfiles médicos de los estudiantes, incluyendo datos como alergias, enfermedades crónicas, medicamentos crónicos y responsables.
Entradas	Término de búsqueda (nombre o carné del estudiante).
Fuente	Base de datos con base a las tablas estudiantes y fichas médicas y manejando con API local.
Salida	Perfiles médicos filtrados con detalles .
Destino	Auxiliares del sistema.
Acción	Búsqueda de estudiantes por nombre o carne, despliegue de fichas médicas completas.
Requisitos	Conexión con la API del sistema, filtrado eficiente.
Precondición	Los estudiantes deben ser registrados por el administrador previamente en el sistema.

Tabla de requerimiento no funcional 1

La tabla de requerimiento no funcional 1 aborda la disponibilidad del sistema, que debe estar operativo durante los días de clase y horas laborales. Esto asegura que auxiliares y personal administrativo tengan acceso al sistema en momentos críticos. Se requiere un servidor con alta disponibilidad y un mantenimiento adecuado.

Requerimiento no funcional Nº1	
Función	Disponibilidad
Descripción	El sistema debe de estar disponibles durante los días de clase durante las horas de trabajo
Entradas	N/A
Fuente	N/A
Salida	N/A
Destino	Auxiliares, administrativos
Acción	Garantizar alta disponibilidad del sistema durante los períodos críticos
Requisitos	Alta disponibilidad del servidor (API)
Precondición	Mantenimiento adecuado del servidor y de la base de datos

3. Aplicación web

Requerimiento funcional Nº1	
Función	Gestión de Medicamentos Permitidos
Descripción	El administrador será capaz de controlar y manejar a los usuarios dentro del sistema
Entradas	Datos del medicamento (nombre, descripción, dosis, etc.)
Fuente	Base de datos con base a las tablas de medicamentos y fichas médicas, manejando con API local
Salida	Lista actualizada de medicamentos permitidos por estudiante
Destino	Administrador del sistema
Acción	Controlar la lista de medicamentos permitidos y asegurarse de que están actualizados según las directrices de salud
Requisitos	Conexión con la API del sistema, filtrado eficiente
Precondición	Los medicamentos deben estar definidos y registrados en el sistema

Requerimiento funcional Nº2	
Función	Gestión de Usuarios
Descripción	El administrador podrá crear, editar o eliminar usuarios, asignando roles y permisos específicos dentro del sistema.
Entradas	Información del usuario (nombre, correo electrónico, contraseña, rol)
Fuente	Base de datos con base a las tablas de usuarios, manejando con API local
Salida	Información actualizada de los usuarios en el sistema
Destino	Administrador del sistema
Acción	Controlar y gestionar todos los usuarios para garantizar el acceso seguro al sistema
Requisitos	Conexión con la API del sistema, filtrado eficiente
Precondición	Los roles de usuario deben estar definidos en el sistema

Requerimiento funcional Nº3	
Función	Gestión de Reportes de Incidentes
Descripción	El administrador podrá visualizar y gestionar todos los reportes de incidentes generados por los auxiliares y responsables familiares
Entradas	Detalles del reporte de incidentes (nombre del estudiante, tipo de incidente, fecha, etc.)
Fuente	Base de datos con base a las tablas de incidentes, manejando con API local
Salida	Listado de reportes de incidentes por estudiante
Destino	Administrador del sistema
Acción	Revisar y tomar decisiones sobre incidentes reportados para garantizar la seguridad de los estudiantes
Requisitos	Conexión con la API del sistema, filtrado eficiente
Precondición	Deben existir reportes de incidentes generados en el sistema

Requerimiento no funcional Nº4	
Función	Gestión de Perfiles Médicos
Descripción	El administrador podrá acceder y gestionar todos los perfiles médicos de los estudiantes, incluyendo alergias y enfermedades registradas
Entradas	Término de búsqueda (nombre o carné del estudiante)
Fuente	Base de datos con base a las tablas de incidentes, manejando con API local
Salida	Detalles de los perfiles médicos de los estudiantes
Destino	Administrador del sistema
Acción	Controlar la información médica de los estudiantes para asegurar la correcta administración de su salud
Requisitos	Que un estudiante tenga un incidente para generar un reporte
Precondición	Los perfiles médicos deben estar creados y registrados en el sistema

Tabla de requerimiento no funcional 1

Requerimiento no funcional Nº1	
Función	Disponibilidad
Descripción	El sistema debe estar disponible y accesible durante los días y horas laborales para el personal administrativo
Entradas	N/A
Fuente	N/A
Salida	N/A
Destino	Administrador y personal administrativo.
Acción	Garantizar que el sistema esté operativo en momentos críticos
Requisitos	Alta disponibilidad del servidor y sistema
Precondición	Mantenimiento adecuado del servidor y de la base de datos

2.3.3 Requerimientos de software

Teniendo en cuenta los lineamientos planteados y los conocimientos adquiridos en el colegio, así como el aprendizaje del equipo de trabajo, se utilizaron las siguientes herramientas y software para complementar el desarrollo del sistema y agilizar el trabajo de auxiliares y administradores

Herramienta de software	Nombre
Tecnologías de Desarrollo	Java, React Native, Flet
Sistema de Gestor de Base de Datos	MySQL
Servidor de Aplicaciones Web	Apache Tomcat 8
Entornos de Desarrollo Integrado(IDE)	Visual Studio Code, Netbeans
Sistema Operativo	Windows, Android
Herramienta para el diseño y modelado de datos	Canva, LucidChart
Framework utilizado para la API	Spring Boot
Cliente HTTP para consumir la API	Axios, Fetch API
Herramienta de conexión con MySQL	XAMPP
Herramienta de oficina	Documentos de Google (Word)

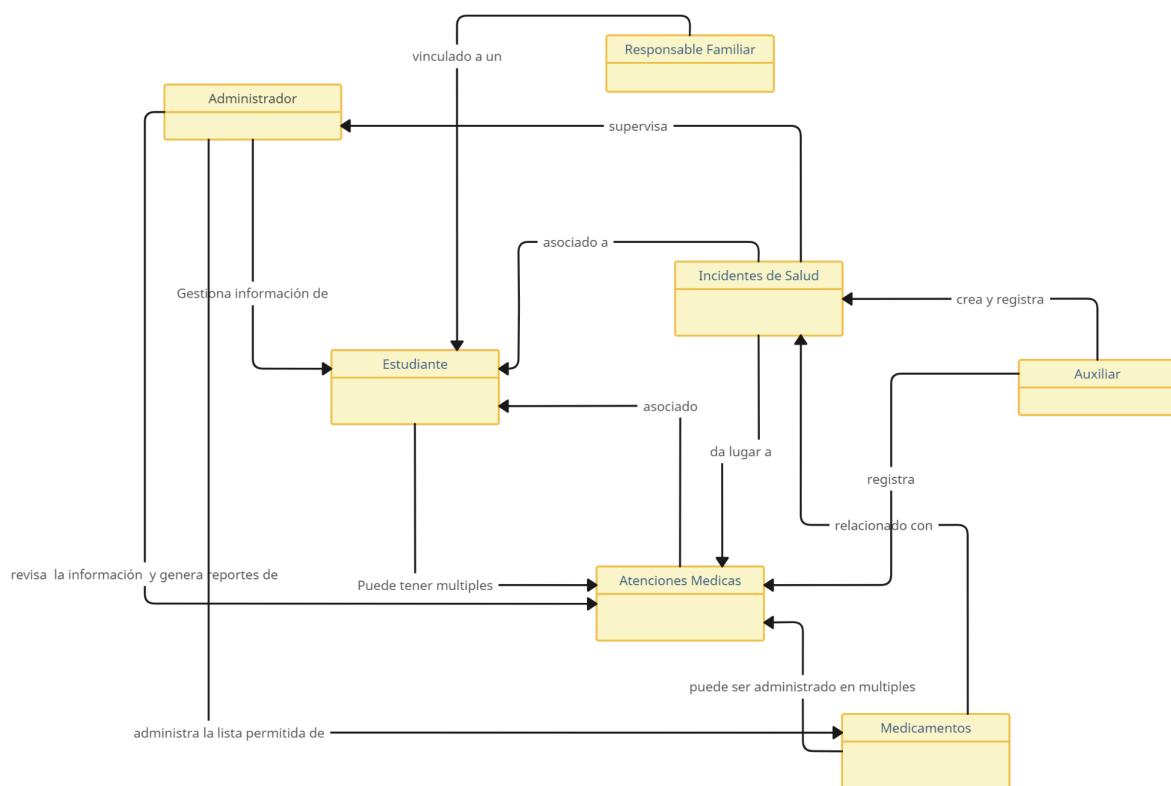
2.3.4 Requerimiento de recursos humano

El equipo de trabajo compuesto por cuatro miembros:

Cada miembro contribuyó activamente al desarrollo del sistema, participó en la documentación del proceso y llevó a cabo pruebas para garantizar su funcionamiento, asegurando así un desarrollo eficiente.

2.4 Modelo de Dominio

Este diagrama ilustra las entidades principales del sistema, así como las acciones que cada una de ellas ejecuta dentro del mismo.



2.5 Casos de uso del sistema

Símbolo	Descripción
Actor 	Actor: es un tipo de usuario del sistema, simbolizado por una figura humana (un monigote). Un usuario es cualquier entidad externa, no necesariamente humano, que interactúa con el sistema.
Caso de Uso 	Caso de uso: es una tarea que se realiza ya sea tras la orden de un actor o bien de la invocación desde otro caso de uso.
Asociación de Comunicación 	Asociación: es el tipo de relación más básica entre un actor y un caso de uso, es cuando el actor interactúa con el sistema para llevar a cabo el caso de uso

2.5.1 Identificación de actores

A continuación, se presenta la lista de los actores identificados y que participarán en el sistema informático de vigilancia estudiantil y logística de atención

No.	Actor	Descripción
1	Administrador	Persona encargada de gestionar el sistema, incluyendo la gestión de incidentes, generación de reportes y administración de las distintas partes que conforman el sistema
2	Auxiliar	Reportar incidentes y consultar las fichas médicas para solventar de la mejor forma dicho incidente.
3	Responsable Familiar	Ver el registro del estudiante al cual se encuentra asociado.
4	Estudiante	Persona que recibe el apoyo por parte de los auxiliares en caso de algún incidente que este mismo presente.

2.5.2 Descripción de casos de uso

A continuación, se describe el estándar establecido para la descripción de los diagramas de caso de uso

Caso de Uso:	Nombre del caso de uso	N°	CDU-00
Actor	Persona o entidad que participa en el caso de uso		
Descripción	Detalles sobre el caso de uso		
Flujo Principal	Secuencia de pasos más común que se lleva a cabo para la ejecución del caso de uso		
Flujo Alternativo	Describe un escenario alternativo al principal		
Precondición	Se detallan las condiciones de acceso en el apartado de precondición		
Validaciones	Se detallan las validaciones que se deben realizar durante la operación		

2.5.3 Casos de Uso

Tabla de casos de uso para aplicación de escritorio

Tabla de caso de uso 1

La tabla de caso de uso 1 detalla el proceso de Iniciar Sesión para auxiliares, permitiendo el acceso a la aplicación con credenciales válidas

Caso de Uso	Iniciar sesión	Nº1	CDU-01
Actor	Auxiliar		
Descripción	Permite a los usuarios auxiliares acceder a la aplicación mediante sus credenciales. Solo los auxiliares podrán ingresar al sistema.		
Flujo Principal	<p>El usuario abre la aplicación y se muestra la pantalla de Login.</p> <p>El usuario introduce su correo electrónico y contraseña.</p> <p>El sistema verifica las credenciales y el rol del usuario.</p> <p>Si las credenciales son válidas y el rol es auxiliar: El usuario es redirigido a la Vista Principal (Home).</p>		
Flujo Alternativo	<p>F1: Si las credenciales son incorrectas, se muestra un mensaje de error: "Credenciales incorrectas".</p> <p>F2: Si el usuario no es auxiliar, se muestra el mensaje: "Acceso denegado: Solo auxiliares pueden acceder."</p>		
Precondición	El usuario debe estar registrado en el sistema con un correo válido y el roleId = 2. La API debe estar disponible para verificar las credenciales.		
Postcondición	Si el inicio de sesión es exitoso, el usuario accede a la Vista Principal .		
Validaciones	<p>Validación del correo y contraseña no vacíos.</p> <p>Validación del roleId = 2 para permitir el acceso.</p>		

Tabla de caso de uso 2

La Tabla de caso de uso 2 describe el proceso de consultar ficha médica. Permite a los auxiliares acceder y visualizar las fichas médicas de los estudiantes desde la vista principal

Caso de Uso	Consultar ficha médica	Nº2	CDU-02
Actor	Auxiliar		
Descripción	Permite al auxiliar acceder a las fichas médicas de los estudiantes.		
Flujo Principal	<p>Desde la Vista Principal, el usuario selecciona la opción "Fichas Médicas".</p> <p>El sistema recupera y muestra las fichas médicas en filas de tres tarjetas.</p> <p>El usuario puede navegar y visualizar las fichas.</p>		
Flujo Alternativo	F1: Si no hay fichas disponibles, se muestra un mensaje: " No hay fichas disponibles ".		
Precondición	<p>El usuario debe haber iniciado sesión como auxiliar.</p> <p>La API debe estar disponible para recuperar las fichas médicas.</p>		
Postcondición	Las fichas médicas se muestran correctamente en pantalla.		
Validaciones	<p>Verificar que la respuesta de la API contenga datos.</p> <p>Validar el formato de los datos antes de mostrarlos en las tarjetas.</p>		

Tabla de caso de uso 3

La tabla de caso de uso 3 detalla el proceso de registrar incidentes. Permite a los auxiliares registrar incidentes relacionados con estudiantes completando un formulario con datos específicos

Caso de Uso	Registrar incidente	Nº3	CDU-03
Actor	Auxiliar		
Descripción	Permite al auxiliar registrar un incidente relacionado con un estudiante.		
Flujo Principal	<p>El usuario selecciona la opción "Registrar Incidente" desde la VistaPrincipal.</p> <p>El usuario completa el formulario con los siguientes datos:</p> <ul style="list-style-type: none"> • Nombre del Estudiante • Fecha del Incidente • Descripción • Tipo de Incidente <p>El usuario presiona el botón "Registrar".</p> <p>El sistema envía la información al servidor para registrar el incidente.</p> <p>Si el registro es exitoso, se muestra un mensaje de confirmación.</p>		
Flujo Alternativo	F1: Si el registro falla, se muestra un mensaje: " Error al registrar el incidente ".		
Precondición	El usuario debe haber iniciado sesión como auxiliar.		
Postcondición	El incidente queda registrado en el sistema.		
Validaciones	<p>Verificar que todos los campos obligatorios estén completos.</p> <p>Validar que el incidente no sea un duplicado antes de registrarlo.</p>		

Tabla de caso de uso 4

La tabla de caso de uso 4 describe el proceso de cerrar sesión. Permite a los auxiliares cerrar su sesión y regresar a la pantalla de Login. Si se presenta un error al cerrar la sesión, se muestra un mensaje de aviso. El usuario debe haber iniciado sesión previamente para realizar esta acción.

Caso de Uso	Cerrar sesión	Nº4	CDU-04
Actor	Auxiliar		
Descripción	Permite al usuario auxiliar cerrar su sesión y volver a la pantalla de Login .		
Flujo Principal	El usuario selecciona la opción " Cerrar Sesión " desde la Vista Principal. El sistema cierra la sesión y redirige al usuario a la pantalla de Login .		
Flujo Alternativo	F1: Si hay algún error al cerrar sesión, se muestra un mensaje: " Error al cerrar sesión ".		
Precondición	El usuario debe haber iniciado sesión previamente.		
Postcondición	La sesión queda cerrada y el sistema muestra la pantalla de Login.		
Validaciones	Verificar que los datos de sesión se limpien correctamente.		

Diagramas de casos de uso aplicación de escritorio

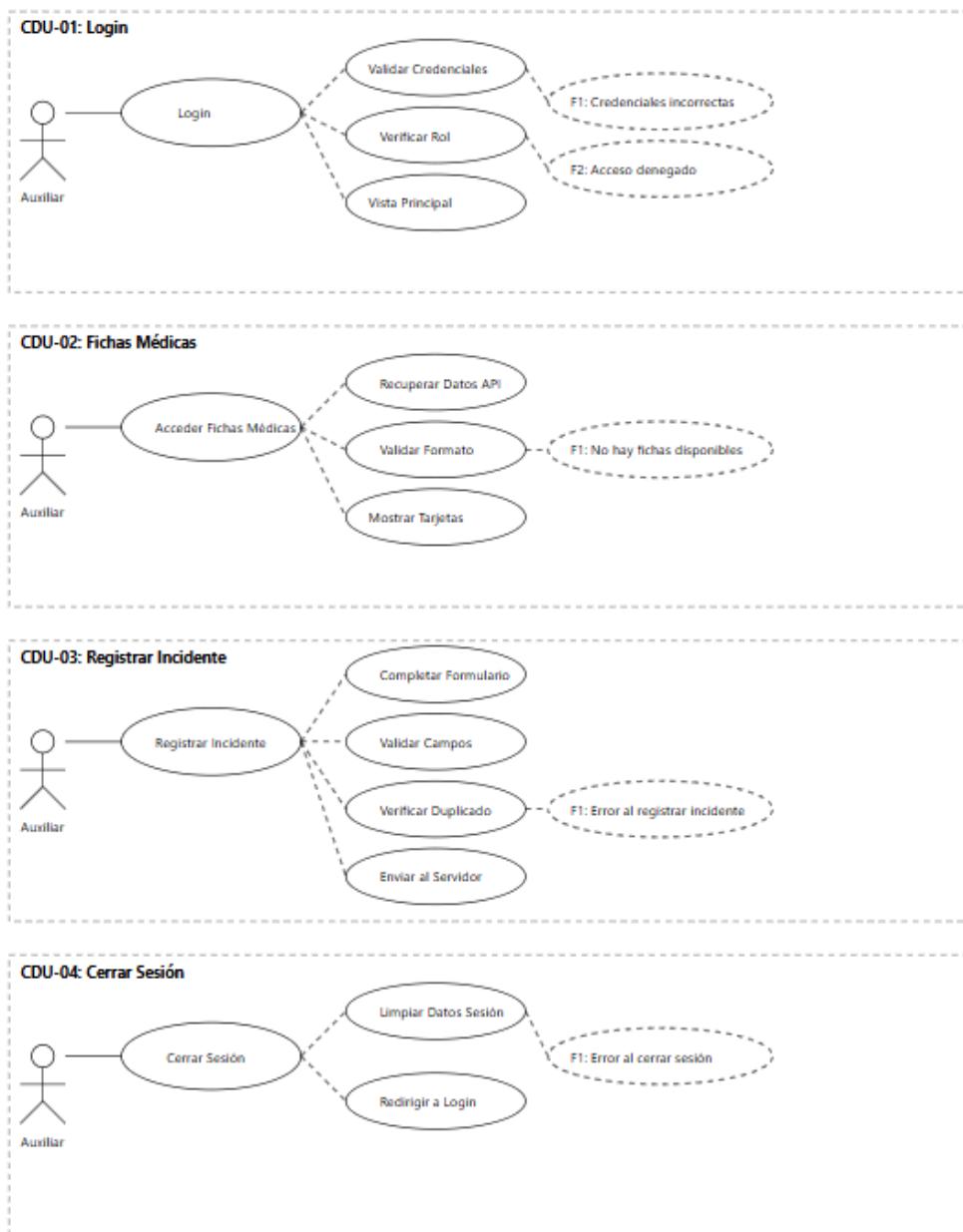


Tabla de casos de uso para aplicación de celular y web

Tabla de caso de uso 1

Permite a los usuarios (Administrador, Auxiliar, Responsable Familiar) ingresar al sistema mediante su correo y contraseña.

Caso de Uso	Iniciar sesión	Nº1	CDU-01
Actor	Administrador, Auxiliar, Responsable Familiar		
Descripción	Permite al usuario iniciar sesión dependiendo de su rol como usuarios en el sistema.		
Flujo Principal	Se mostrará una vista al usuario “Iniciar Sesión” donde podrá escribir su gmail y su contraseña para ingresar al sistema de “VELARRUPE”.		
Flujo Alternativo	F1: Si los datos no son correctos, mostrará un error de (Datos incorrectos). F2: Si el error persiste, avisar al administrador para verificar las credenciales de su usuarios correspondientes.		
Precondición	El usuario debe tener cuenta registrada.		
Postcondición	Una vez el usuario tenga su sesión iniciada, dependiendo de su rol como usuario tendrá los apartados correspondientes ya sea “Administrador, Auxiliar, Responsable Familiar”.		
Validaciones	Verificar que los datos obligatorios como los son el “gmail, contraseña” están siendo colocados correctamente.		

Tabla de caso de uso 2

Permite al administrador realizar diversas acciones en el sistema, como gestionar estudiantes, usuarios y fichas médicas.

Caso de Uso	Vista Administrador	Nº2	CDU-02
Actor	Administrador		
Descripción	<p>Permite al administrador realizar diferentes acciones dentro del sistema, ejemplo de estos apartados son con las siguientes funciones:</p> <ul style="list-style-type: none"> • Estudiantes(Agregar, Ver, Editar, Eliminar) • Usuarios(Agregar, Ver, Editar, Eliminar) • Fichas Médicas(Agregar, Ver, Editar, Eliminar) • Informar Incidentes de los estudiantes • Vincular responsables familiares con su estudiante • Permitir medicamentos a los estudiantes • Aceptar solicitudes de retiro para el estudiante. 		
Flujo Principal	<p>El administrador tendrá su propio apartado para manejar todo el sistema y la base de datos por medio de diferentes apartados, esto le da todo privilegio como administrador ante todo el sistema.</p>		
Flujo Alternativo	<p>F1: En caso que alguna función de todas las posibles en el panel de administrador, saldrá un mensaje de 'No se ha encontrado la ruta de la vista' si esto no sucede, todo el panel de administrador estará funcionando correctamente".</p>		
Precondición	<p>El usuario debe tener una cuenta como administrador en el sistema, si no este jamás podrá ingresar a este apartado exclusivo.</p>		
Postcondición	<p>Previamente se necesita haber iniciado sesión con los datos correctos y como usuarios administrador.</p>		
Validaciones	<p>Verificar si el rol del usuario que está ingresando es de tipo administrador.</p>		

Tabla de caso de uso 3

El administrador registra estudiantes en el sistema completando campos obligatorios como nombre, apellido, número de carnet y fecha de nacimiento.

Caso de Uso	Registrar Estudiantes	Nº3	CDU-03
Actor	Administrador		
Descripción	El administrador tiene un apartado donde podrá registrar a todos los estudiantes de la institución “Colegio Español Padre Arrupe” esto para posteriormente guardar todos sus registros en el sistema y que pueda tener todos historial de incidentes dentro de la institución.		
Flujo Principal	Agregar estudiante completando todos los campos obligatorios, en este caso los datos son los siguientes (nombre, apellido, n°carnet, fecha de nacimiento) con estos datos el estudiante podrá estar registrado dentro de nuestro sistema.		
Flujo Alternativo	<p>F1: Si al campo de “nombre” se intenta colocar un número, este no lo aceptará y pedirá que lo vuelva a colocar.</p> <p>F2. Similar pasa con el campo “Apellido” el campo detectará si hay número y volverá a pedir el apellido hasta que se coloque sin números.</p> <p>F3. En el campo “n°carnet” del estudiante, lo que pedirá es que se coloque la primera letra del nombre y la primera del apellido en mayúsculas para luego posteriormente colocar 6 números.</p> <p>F4. La fecha debe ser hace más de 4 años, ya que es lo permitido para ingresar a un estudiante en la institución</p>		
Precondición	El administrador debe ser cauteloso para poder ingresar a todos los estudiantes con sus respectivos datos correctos.		
Postcondición	Una vez registrados todos los datos del estudiante procederán a guardarse en la base de datos.		
Validaciones	Los datos de los estudiantes para el registro deben ser 100% correctos para luego más adelante evitar algún cambio.		

Tabla de caso de uso 4

Permite al administrador modificar los datos de los estudiantes. Se deben cumplir los requisitos de entrada para evitar errores.

Caso de Uso	Editar Estudiante	Nº4	CDU-04
Actor	Administrador		
Descripción	El administrador tiene un apartado donde podrá editar a cualquier estudiante de la institución “Colegio Español Padre Arrupe” esto por si en algún momento el usuario requiera alguna modificación dada por altos cargos de la institución y se debe hacer.		
Flujo Principal	Al modificar algún estudiante, debes tomar en cuenta siempre colocar los datos requeridos de manera correcta, si no se hace el sistema lo detectará y te pedirá que coloques los campos con sus respectivos requerimientos .		
Flujo Alternativo	<p>F1: Si al campo de “nombre” se intenta colocar un número, este no lo aceptará y pedirá que lo vuelva a colocar.</p> <p>F2. Similar pasa con el campo “Apellido” el campo detectará si hay número y volverá a pedir el apellido hasta que se coloque sin números.</p> <p>F3. En el campo “nºcarnet” del estudiante, lo que pedirá es que se coloque la primera letra del nombre y la primera del apellido en mayúsculas para luego posteriormente colocar 6 números.</p> <p>F4. La fecha debe ser hace más de 4 años, ya que es lo permitido para ingresar a un estudiante en la institución</p>		
Precondición	El administrador debe ser cauteloso para poder editar a los estudiantes con sus respectivos datos correctos.		
Postcondición	Una vez editado el estudiante por sus altos cargos los datos procederán a actualizarse en la base de datos.		
Validaciones	Estar seguros al momento de editar algún estudiante, ya que esto solo se debe ser al menos que sea de motivo de fuerza mayor al realizarlo.		

Tabla de caso de uso 5

El administrador puede crear nuevos usuarios con roles (Administrador, Auxiliar, Responsable Familiar) y sus datos correspondientes. Se valida la información ingresada para asegurar la correcta creación del usuario en el sistema.

Caso de Uso	Agregar Usuario	Nº5	CDU-05
Actor	Administrador		
Descripción	El administrador tiene un apartado donde podrá agregar cualquier usuario para la institución “Colegio Español Padre Arrupe” debe cumplir uno de estos roles “Administrador, Auxiliar, Responsable Familiar” para tener acceso dependiendo el tipo de rol al sistema.		
Flujo Principal	El administrador tendrá la posibilidad de crear un nuevo usuario, si este lo amerita para poder cumplir sus funciones dentro de nuestro sistema con sus datos correspondientes “nombre, email, contraseña, teléfono y rol” .		
Flujo Alternativo	<p>F1: El nombre no debe contener ningún número ni signos, solo letras.</p> <p>F2. Campo email debe tener una terminación de “@gmail.com” para poder ser registrado por un correo electrónico.</p> <p>F3. El número de teléfono debe de ir con este formato “****-****” para poder ingresarlo.</p> <p>F4. Especificar que tipo de rol tendrá el usuario, en este caso están los tres “Admin, Auxiliar, Responsable Familiar”</p>		
Precondición	El administrador debe tener el derecho para poder crear un nuevo usuario.		
Postcondición	Una vez registrado el nuevo usuario, tendrá la posibilidad de poder ingresar al sistema.		
Validaciones	Debemos de tener seguridad con los nuevos usuarios que entren al sistema.		

Tabla de caso de uso 6

Permite al administrador gestionar usuarios (editar, eliminar, ver) dentro del sistema. Se valida que la información proporcionada cumpla con los requisitos establecidos. Se requiere tener el derecho de control sobre los usuarios.

Caso de Uso	Controlar Usuario	Nº6	CDU-06
Actor	Administrador		
Descripción	El administrador tiene un apartado donde podrá controlar cualquier usuario para la institución “Colegio Español Padre Arrupe” debe cumplir uno de estos roles “Administrador, Auxiliar, Responsable Familiar” para tener acceso dependiendo el tipo de rol al sistema.		
Flujo Principal	Permite al administrador realizar diferentes acciones dentro del sistema hacia los usuarios que son las siguientes: <ul style="list-style-type: none"> • Editar Usuario. • Eliminar Usuario. • Ver Usuario 		
Flujo Alternativo	F1: El nombre no debe contener ningún número ni signos, solo letras. F2: Campo email debe tener una terminación de “@gmail.com” para poder ser registrado por un correo electrónico. F3: El número de teléfono debe de ir con este formato “****-****” para poder ingresarlo. F4: Especificar que tipo de rol tendrá el usuario, en este caso están los tres “Admin, Auxiliar, Responsable Familiar”		
Precondición	El administrador debe tener el derecho para poder controlar todos los usuarios dentro del sistema.		
Postcondición	Poder controlar a los usuarios por si en algún momento se necesita alguna modificación.		
Validaciones	Debemos de tener seguridad controlar a todos los usuarios que entren al sistema.		

Tabla de caso de uso 7

El administrador gestiona los medicamentos permitidos para estudiantes, basándose en la autorización de sus responsables familiares. Se requiere asegurarse de que solo se asignen medicamentos autorizados.

Caso de Uso	Controlar Medicamentos Permitidos	Nº7	CDU-07
Actor	Administrador		
Descripción	El administrador tiene un apartado donde podrá manejar los medicamentos permitidos para los estudiantes dentro de la institución “Colegio Español Padre Arrupe” esto debe ser con muy cuidadoso ya que los responsables familiares son los que les autorizaron a la institución permitir los medicamentos.		
Flujo Principal	El administrador tienen el control de acceder a medicamentos al estudiante mediante las opciones de su respectivo responsable familiar: <ul style="list-style-type: none"> ● Ver estudiantes. ● Ver medicamentos. ● Permitirle medicamentos a los estudiantes 		
Flujo Alternativo	F1: El estudiante solo debe tener registro de las pastillas permitidas por su responsable familiar. F2. No se le podrá asignar algún medicamento que su responsable no haya autorizado en la ficha del estudiante.		
Precondición	El administrador debe estar completamente seguro al momento de asignarle los medicamentos permitidos al estudiante.		
Postcondición	Una vez asignando los medicamentos al estudiante, el colegio se encargará de cumplir en caso de alguna emergencia.		
Validaciones	Si el responsable no ha autorizado ningún medicamento dentro de la institución, al estudiante no se le permitirá dar medicamentos.		

Tabla de caso de uso 8

El administrador gestiona las fichas médicas de los estudiantes, que contienen información sensible sobre alergias y enfermedades.

Caso de Uso	Fichas Médicas	Nº8	CDU-08
Actor	Administrador		
Descripción	El administrador tendrá la capacidad de manejar las fichas médicas de los estudiantes de la institución, ya que esta tiene información delicada como lo son las alergias, medicamentos crónicos o enfermedades crónicas .		
Flujo Principal	El administrador tienen el control de manejar fichas médicas de los estudiantes mediante las opciones: <ul style="list-style-type: none"> ● Ver fichas médicas. ● Editar fichas médicas. ● Eliminar fichas médicas. 		
Flujo Alternativo	F1: El estudiante tiene su propia ficha médica la cual el administrador será encargado de registrarla en el sistema. F2. El administrador debe estar muy seguro al controlar la ficha médica de los estudiantes.		
Precondición	En la ficha estudiantil que se entrega a principio del año, se entrega un apartado médico, ese se colocará en este apartado.		
Postcondición	Cuando el estudiante ya tenga su ficha médica en el sistema, el administrador será el encargado de manejarlas todas y cada una de ellas.		
Validaciones	La ficha médica condiciona que el auxiliar quien es encargado de reportar los incidentes, la pueda ver y pueda verificar si el estudiante padece de alguna alergia o enfermedad crónica.		

Tabla de caso de uso 9

El administrador acepta las solicitudes de retiro de estudiantes en caso de malestar y así el responsable familiar pueda venirlo a recoger

Caso de Uso	Aceptar solicitud de retiro	Nº9	CDU-09
Actor	Administrador		
Descripción	El administrador tiene la capacidad de aceptar las solicitudes de retiro de los estudiantes dentro de la institución, esto se debe a que el estudiante puede sentirse muy mal y decide ir a casa pero antes debe ser aprobada la solicitud por el administrador.		
Flujo Principal	El administrador maneja las solicitudes de retiro de los estudiantes para poder aprobarlas y que el estudiante se pueda retirar: <ul style="list-style-type: none"> ● Ver solicitudes pendientes de retiro. ● Aceptar solicitudes pendientes de retiro. ● Informar a los familiares responsables del estudiante para poder ir a retirar su hij@. 		
Flujo Alternativo	F1: En este caso no hay opciones de error ya que solo podrán por medio de un botón aceptar las solicitudes.		
Precondición	Debe informar a los responsables familiares del estudiante que desea retirar para que el administrador pueda aceptar la solicitud.		
Postcondición	Una vez aceptada la petición de retiro por parte del administrador, el estudiante podrá retirarse sin ningún problema.		
Validaciones	Si se acepta la solicitud de retiro, el estado pasa de “Pendiente” a “Aceptada” para que el responsable familiar pueda ir a recoger a su hij@ sin ningún problema.		

Tabla de caso de uso 10

El responsable familiar visualiza incidentes registrados de su estudiante en tiempo real

Caso de Uso	Ver información de incidentes del estudiante	Nº10	CDU-10
Actor	Responsable familiar		
Descripción	Como responsable familiar sólo tendrá la capacidad de visualizar los incidentes que le puedan pasar a su estudiante "hij@" dentro de la institución en tiempo real, esto servirá para mantener informado al responsable de lo que pueda sucederle a su estudiante dentro de las instalaciones .		
Flujo Principal	<p>El responsable podrá verificar el nombre de su estudiante y como información los incidentes que se hayan registrado a la base de datos con su nombre, donde en la tarjeta aparecerá:</p> <ul style="list-style-type: none"> • Nombre del estudiantes. • Incidente que le sucedió. • Fecha en la que sucedió. • Si la solicitud de retiro del estudiante está pendiente o si el estudiante decidió quedarse en la institución 		
Flujo Alternativo	F1: Si no aparece al responsable familiar no le aparece toda la información de su estudiante y su incidentes, debe informar a algún administrador para verificar cuales es el problema.		
Precondición	Debe tener una cuenta como responsable familiar y un estudiante vinculado para poder ver toda la información.		
Postcondición	Si ya tienes una cuenta como responsable familiar y un estudiante vinculado a tu cuenta, les aparecerá todos y cada uno de los incidentes que le puedan pasar al estudiante dentro de la institución.		
Validaciones	El responsable podrá tener de 1 a más estudiantes vinculados a la cuenta ya que pueden tener a más de un estudiante(hij@) cursando en el Colegio Español Padre Arrupe .		

Tabla de caso de uso 11

El auxiliar reporta incidentes que ocurren a los estudiantes, incluyendo detalles de salud.

Caso de Uso	Reportar información de incidentes del estudiante	N°11	CDU-11
Actor	Auxiliar		
Descripción	Como auxiliar tendrá la posibilidad de reportar incidentes y especificar que es lo que le sucedió al estudiante y llevar el control de todos esos incidentes ya que ese reporte también será enviado al familiar .		
Flujo Principal	<p>El auxiliar podrá reportar el incidente de cada estudiante y tendrá la capacidad de poder visualizar la ficha médica y los medicamentos permitidos para cada estudiante y llevar los primeros auxilios como corresponde, de forma rápida y concisa :</p> <ul style="list-style-type: none"> ● Buscar al estudiante por nombre o carnet. ● Reportar exactamente cual es el incidente. ● Revisar la ficha médica. ● Revisar los medicamentos autorizados por el responsable familiar para llevar a cabo los primeros auxilios de forma rápida. 		
Flujo Alternativo	F1: El auxiliar debe llenar los datos de manera correcta, ya que si no especifica cuál es el incidente, no lo dejará reportar el incidente del estudiante.		
Precondición	Preguntarle al estudiante que es lo que le sucede y si desea irse a casa para informar al cliente la solicitud.		
Postcondición	Una vez el reporte es enviado, directamente le llegará la información de lo sucedido al responsable familiar para que esté informado a tiempo real.		
Validaciones	Se muestran 4 incidentes frecuentes: "Dolor de cabeza, estómago y cuerpo" y "Náuseas" si no es ninguno de estos el auxiliar podrá especificar cual es el incidente por medio de describir qué es lo que le sucedió específicamente.		

2.5.4 Diagramas de casos uso para aplicación móvil y web

Diagrama 1 de caso de uso

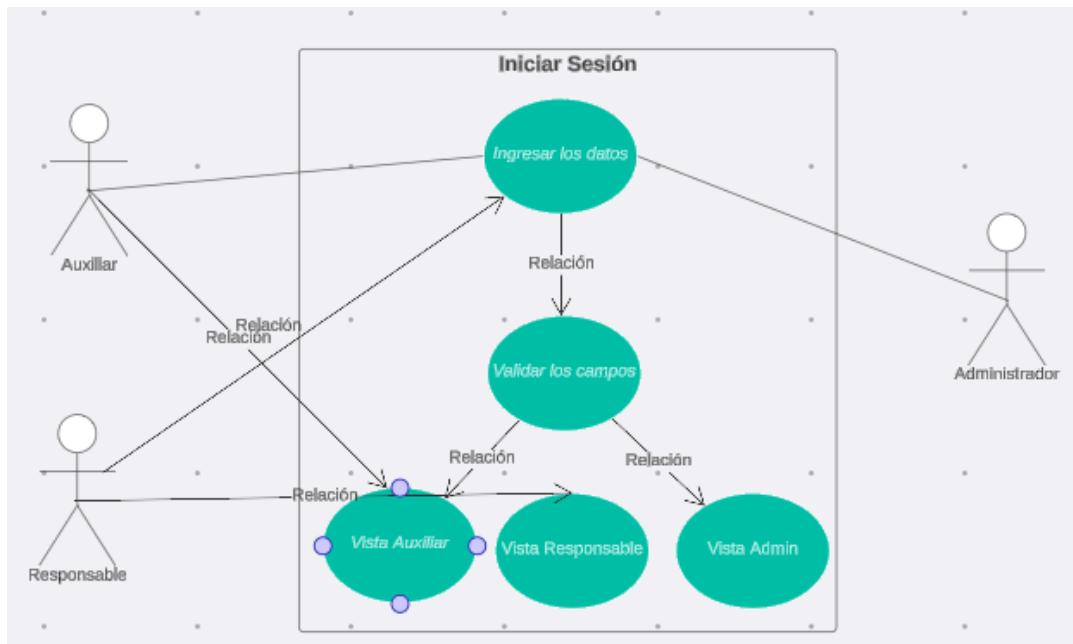


Diagrama 2 de caso de uso

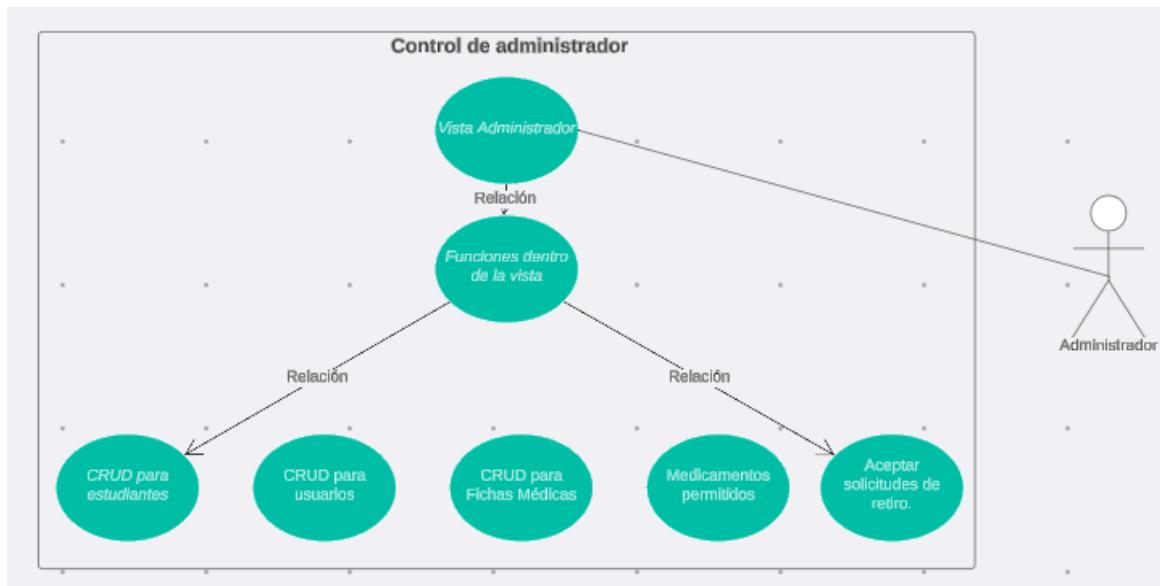


Diagrama 3 de caso de uso

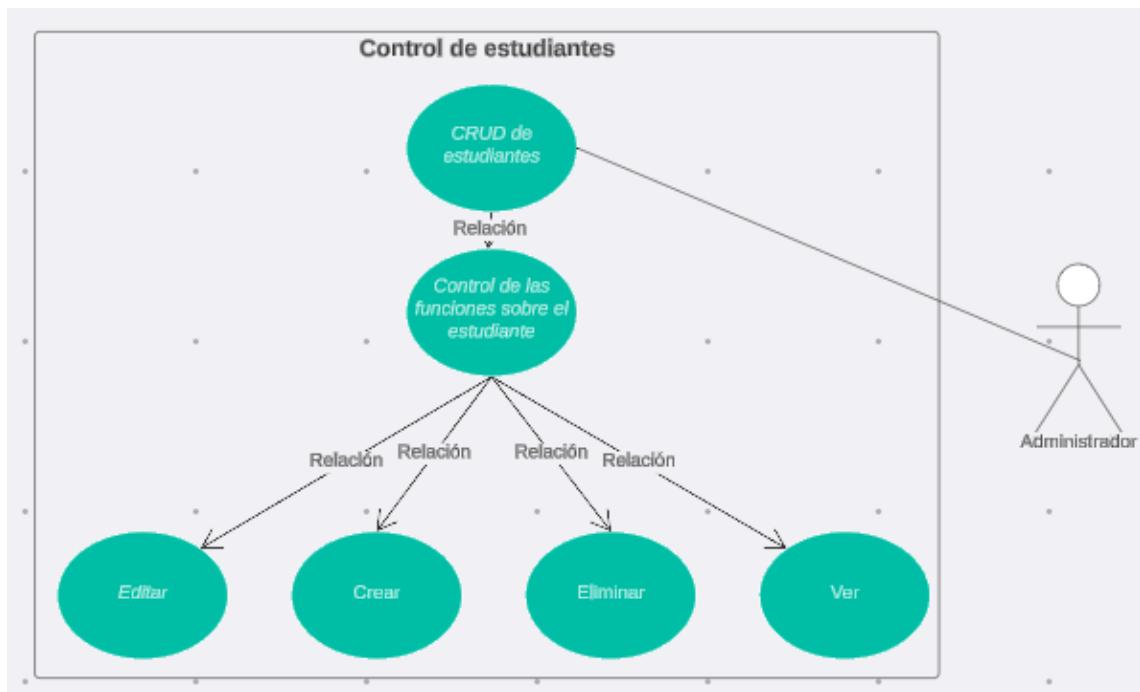


Diagrama 4 de casos de uso

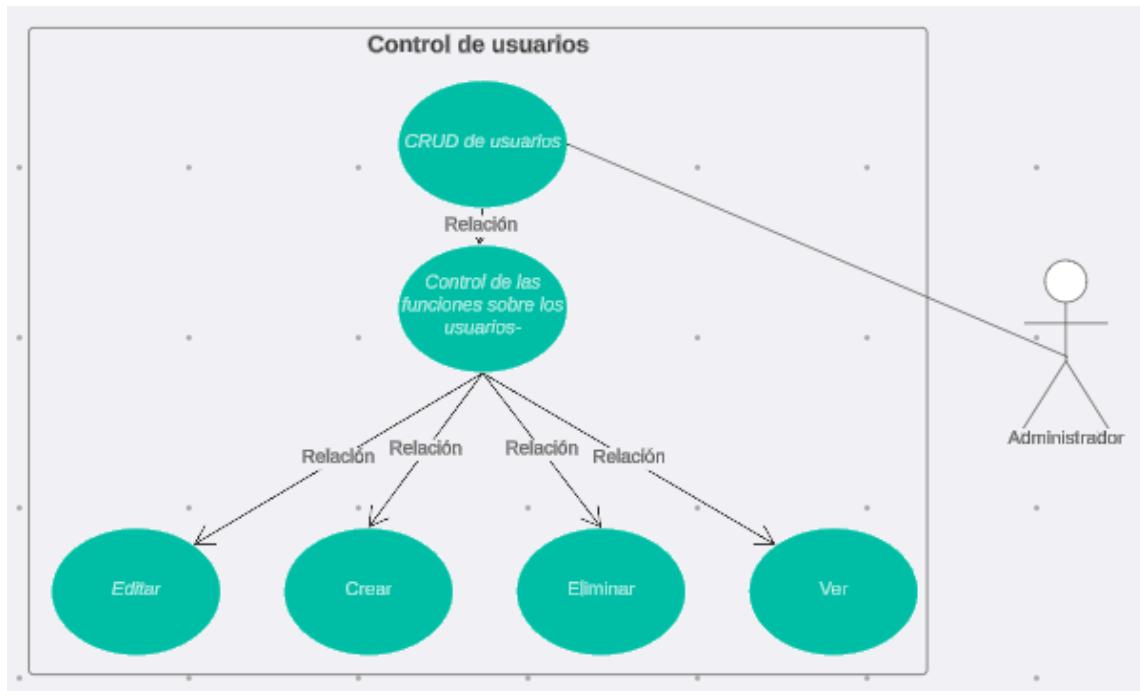


Diagrama 5 de caso de uso

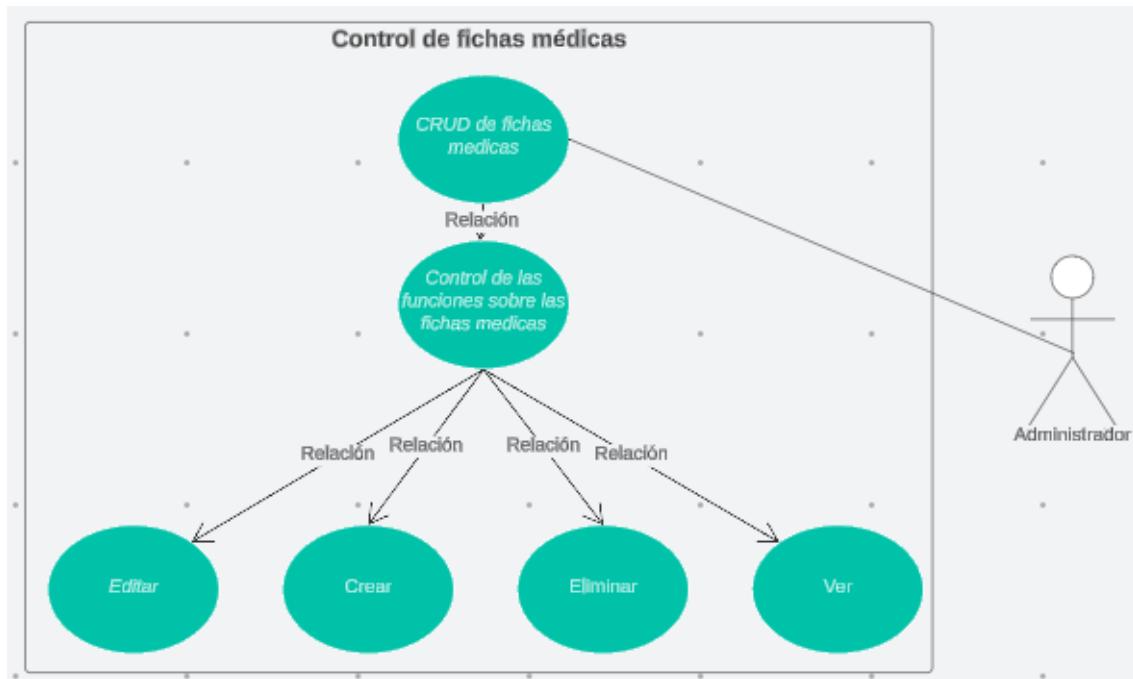


Diagrama 6 de caso de uso

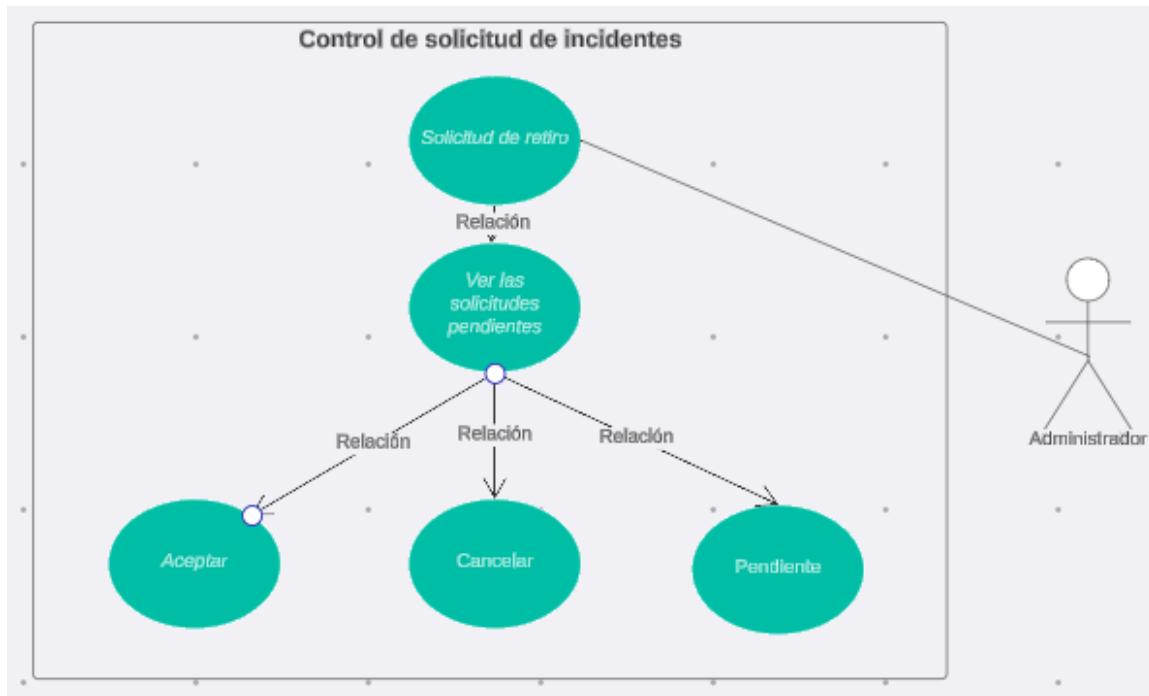


Diagrama 7 de caso de uso

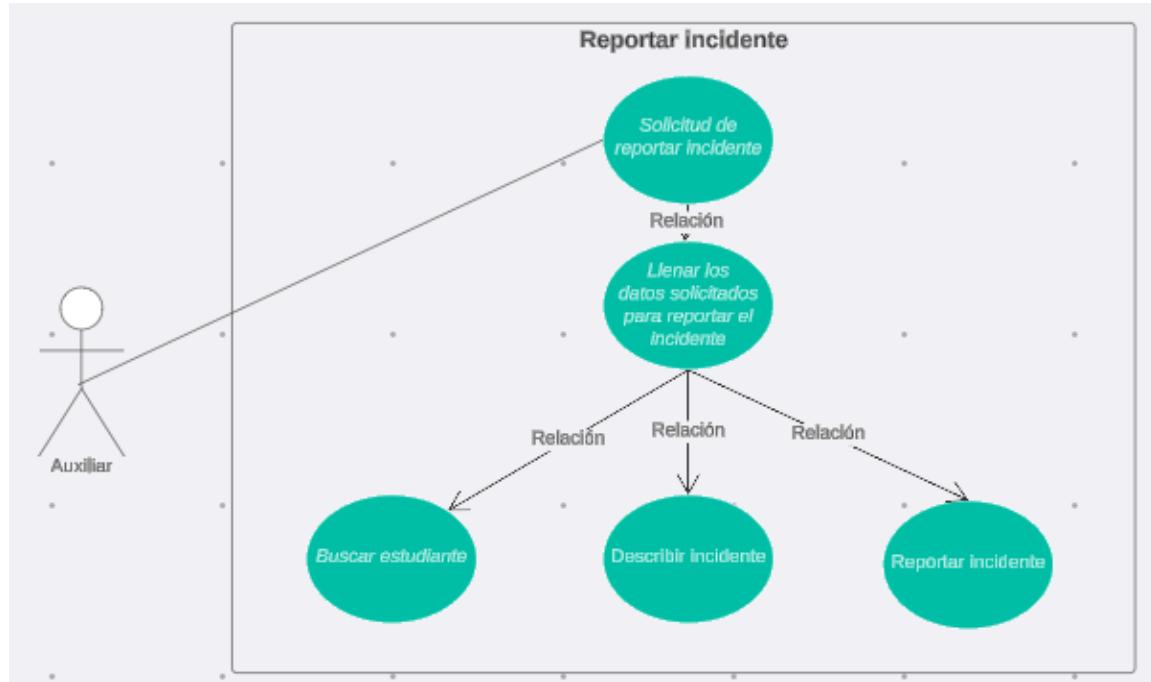
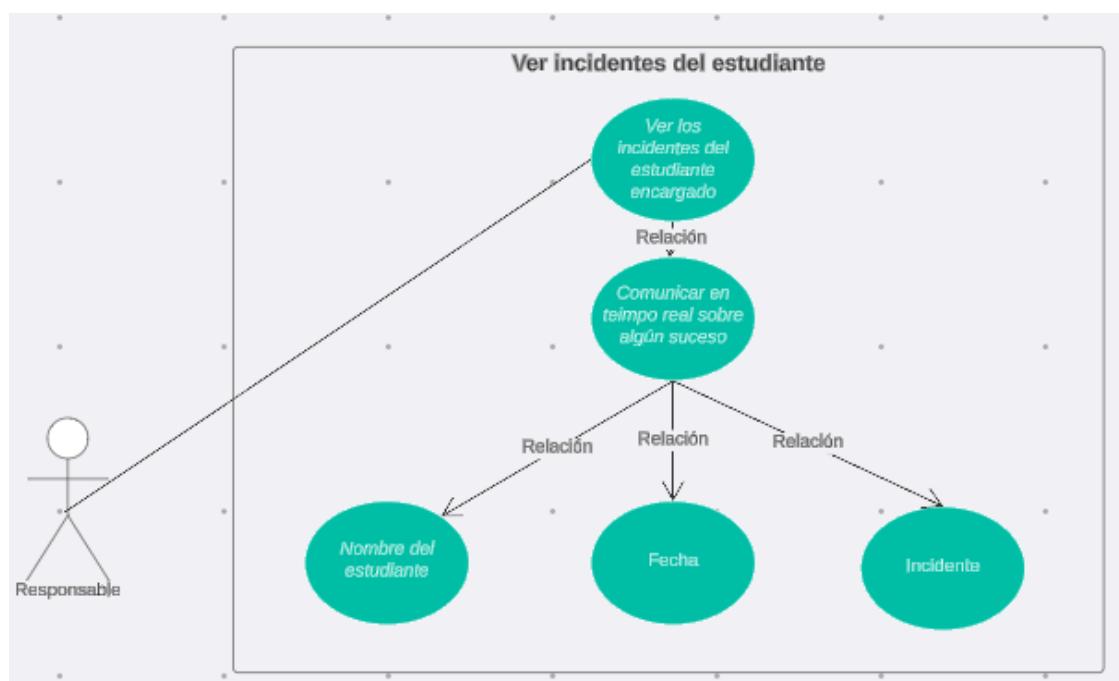


Diagrama 8 de caso de uso



2.6 Historias de usuario

Se seguirá el siguiente formato para la documentación de las historias de usuario con las 4 entidades principales

- 1- Administrador
- 2- Auxiliar
- 3- Estudiante
- 4- Responsable familiar

Historia de usuario de Administrador

Actor	Descripción
Administrador	Poder crear y editar perfiles de estudiantes
	Poder gestionar fichas médicas de los estudiantes
	Poder crear y asignar responsables familiares a los estudiantes
	Poder aprobar o rechazar solicitudes de retiro de estudiantes.
	Poder generar y descargar PDFs de las fichas médicas

Historia de usuario de auxiliar

Actor	Descripción
Auxiliar	Poder acceder a la ficha médica de un estudiante mediante su carné
	Poder gestionar fichas médicas de los estudiantes
	Poder realizar y registrar datos sobre el estado del estudiante
	Poder registrar atenciones médicas, seleccionando enfermedades preestablecidas
	Poder verificar medicamentos autorizados para cada estudiante
	Poder solicitar el retiro de un estudiante

Historia de usuario de responsable familiar

Actor	Descripción
Responsable Familiar	Como responsable familiar quiero poder ver el registro de medicamentos aplicados a los estudiantes
	Como responsable familiar quiero poder acceder al historial de atenciones médicas de los estudiantes

Historia de usuario de estudiante

Actor	Descripción
Estudiante	Como estudiante quiero poder expresar al personal de auxiliar cuando sienta algún malestar o presente alguna situación de riesgo, para así recibir atención rápidamente

2.7 Estándares de diseño

En este apartado se muestra los estándares que será utilizado para la creación de los sistemas

Interfaz Centrada y Espaciada

El contenido principal, como los títulos y los formularios, debe estar centrado horizontalmente.

Utiliza un espaciado entre los componentes para mejorar la legibilidad y navegación.

Tipografía

Usa tipografía clara y consistente, con tamaños variables para resaltar información importante.

Accesibilidad y Claridad:

Textos legibles, con suficiente contraste de colores entre el fondo y las fuentes.

2.8 Paleta de colores

Tomando en cuenta la sugerencia de la licenciada Liliana Landaverde se utilizó la paleta de colores oficial de la Fundación Padre Arrupe destacando como el tono principal el color azul

Primario: "#1A4B8C" Color azul

Secundario: "#F5F7FA" Color blanco con tono gris muy suaves para el contraste

3. CONSTRUCCIÓN DE LA MARCA:

colores

Se mantiene el tono azul como color identificativo de la Fundación Padre Arrupe, al que se unen una gama de colores secundarios para complementar en sus usos.

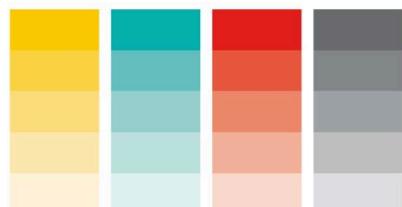
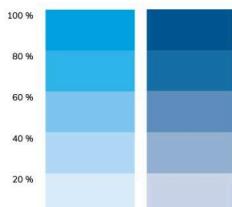
Colores principales

Pantone 299C	Pantone 7691C
Pantone 299CP	Pantone 7691CP
CMYK: 86c 8m	CMYK: 100c 36m 40k
RGB: 162g 225b	RGB: 88g 146b
HEX: #00A2E1	HEX: #006198

Colores secundarios

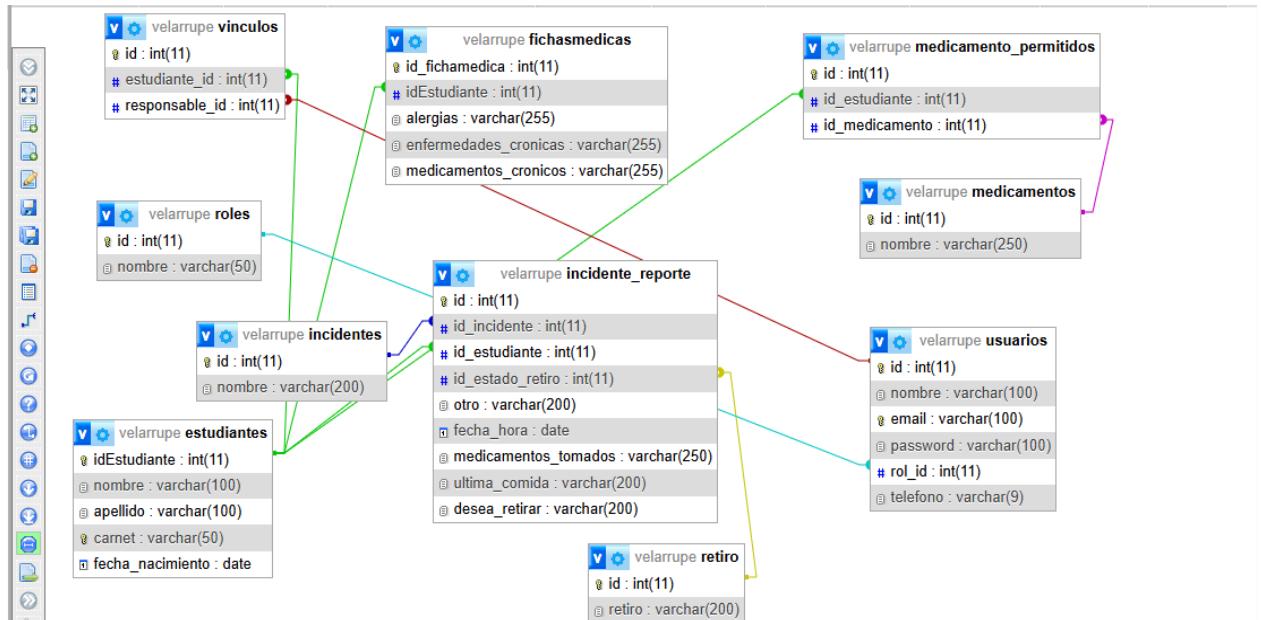
Pantone 7406C	Pantone 3262C	Pantone 1795C	Pantone 424C
Pantone 7406CP	Pantone 3262P	Pantone 1795CP	Pantone 424CP
CMYK: 20m 100y 2k	CMYK: 58c 35y	CMYK: 0c 87m 85y 0k	CMYK: 30c 20m 19y 58k
RGB: 243r 196g	RGB: 111r 194g 182b	RGB: 216r 40g 47b	RGB: 105r 109g 111b
HEX: #F3C400	HEX: #69c9ba	HEX: #D8282F	HEX: #717273

Matices



2.9 Diseño de la Base de Datos

La base de datos está diseñada siguiendo el modelo relacional, utilizando MySQL como sistema gestor



2.9.1 Estructura de tablas

1- Tabla estudiantes

Descripción : Almacena la información básica de los estudiantes

Campos	Tipo	Descripción	Restricciones
idEstudiante	INT(11)	Identificador único del estudiante	PK, AUTO_INCREMENT
nombre	VARCHAR(100)	Nombre del estudiante	NOT NULL
apellido	VARCHAR(100)	Apellido del estudiante	NOT NULL
carnet	VARCHAR(50)	Número de identificación único	NOT NULL, UNIQUE
fecha_nacimiento	DATE	Fecha de nacimiento	NOT NULL

2- Tabla fichas médicas

Descripción : Almacena la información médica relevante de cada estudiante

Campos	Tipo	Descripción	Restricciones
id_fichamedica	INT(11)	Identificador único de la ficha	PK, AUTO_INCREMENT
idEstudiante	INT(11)	Nombre del estudiante	FK, NOT NULL
alergias	VARCHAR(255)	Apellido del estudiante	NULL
enfermedades_cronicas	VARCHAR(255)	Lista de enfermedades crónicas	NULL
medicamentos_cronicos	VARCHAR(255)	Lista de medicamentos de uso regular	NULL

3- Tabla usuarios

Descripción : Gestiona los usuarios del sistema (auxiliares, administradores y responsables familiares)

Campos	Tipo	Descripción	Restricciones
id	INT(11)	Identificador único del usuario	PK, AUTO_INCREMENT
nombre	VARCHAR(100)	Nombre completo del usuario	NOT NULL
email	VARCHAR(100)	Correo electrónico del usuario	NOT NULL, UNIQUE
password	VARCHAR(100)	Contraseña	NOT NULL
rol_id	INT(11)	Identificador del rol asignado	FK
telefono	VARCHAR(9)	Número de teléfono de contacto	NULL

4- Tabla roles

Descripción : Define los roles y permisos en el sistema

Campos	Tipo	Descripción	Restricciones
id	INT(11)	Identificador único del rol	PK, AUTO_INCREMENT
nombre	VARCHAR(50)	Nombre del rol	NOT NULL

5- Tabla vínculos

Descripción : Permite realizar la vinculación del responsable familiar con el estudiante

Campos	Tipo	Descripción	Restricciones
id	INT(11)	Identificador único del vínculo	PK, AUTO_INCREMENT
estudiante_id	INT(11)	Referencia al estudiante	NOT NULL
responsable_id	INT(11)	Referencia al usuario responsable	FK, NOT NULL

6- Tabla incidentes

Descripción : Catálogo de tipos de incidentes más comunes

Campos	Tipo	Descripción	Restricciones
id	INT(11)	Identificador único del vínculo	PK, AUTO_INCREMENT
nombre	VARCHAR(200)	Descripción del tipo de incidente	NULL

7- Tabla de retiro

Descripción : Permite decidir si el estudiante desea retirarse o seguir en la institución

Campos	Tipo	Descripción	Restricciones
id	INT(11)	Identificador único del estado de retiro	PK, AUTO_INCREMENT
retiro	VARCHAR(200)	Descripción del estado de retiro	NULL

8- Tabla incidente reporte

Descripción: Registro detallado de incidentes que presenten los estudiantes

Campos	Tipo	Descripción	Restricciones
id	INT(11)	Identificador único del reporte	PK, AUTO_INCREMENT
desea_retirar	VARCHAR(200)	Indica si se solicita retiro	NULL
fecha_hora	DATE	Fecha y hora del incidente	NULL
id_estado_retiro	INT(11)	Estado del retiro	FK
id_estudiante	INT(11)	Estudiante involucrado	FK
id_incidente	INT(11)	Tipo de incidente	FK
medicamentos_tomados	VARCHAR(250)	Medicamentos administrados	NULL
otro	VARCHAR(200)	Observaciones adicionales	NULL
ultima_comida	VARCHAR(200)	Registro de última ingesta de alimentos	NULL

9- Tabla medicamentos

Descripción: Lista de medicamentos que el colegio puede administrar a los estudiantes

Campos	Tipo	Descripción	Restricciones
id	INT(11)	Identificador único del medicamento	PK, AUTO_INCREMENT
nombre	VARCHAR(250)	Nombre del medicamento	NULL

10- Tabla medicamentos permitidos

Descripción: Registro de medicamentos autorizados para el estudiante

Campos	Tipo	Descripción	Restricciones
id	INT(11)	Identificador único del permiso	PK, AUTO_INCREMENT
id_estudiante	INT(11)	Referencia al estudiante	FK
id_medicamento	INT(11)	Referencia al medicamento	FK

CAPÍTULO III

Capítulo III: Construcción e implementación del sistema.

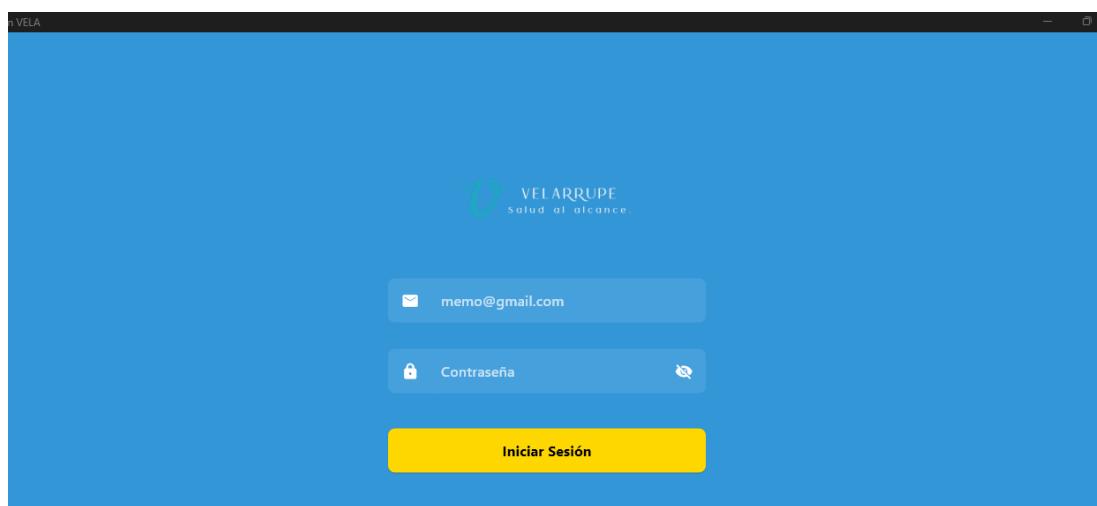
3.1 Detalles de sistemas

Aplicación de escritorio

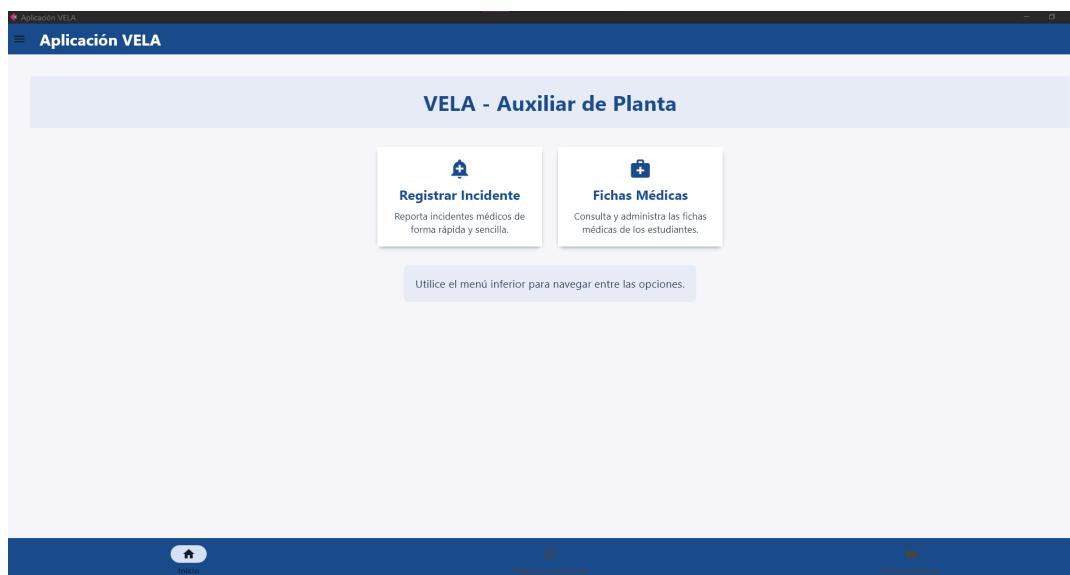
La aplicación de escritorio se enfocó en la vista de auxiliar, el cual puede agregar incidentes y revisar los perfiles médicos. A continuación se presentan más detalles

Inicio de Sesión

Inicio de sesión para ingresar credenciales de auxiliar del sistema de escritorio



Menú principal con barra de navegación que lleva al inicio, a registrar un incidente, o a ver el perfil médico



Ver el perfil médico de un estudiante, ingresar el nombre o el carnet

Aplicación VELA

Buscar Fichas Médicas

Buscar por nombre o carnet

ID: 1

Héctor Raúl Sorto Parada
Carnet: SP220153

Información Médica
Alergias: Polvo, césped, pelo de gatos y perros
Enfermedades Crónicas: Ninguna
Medicamentos Permitidos: Acetaminofén, Cloferinamina

Contacto de Emergencia
Responsable: María Celia
Teléfono: 7933-1951

Home Register Incident Fichas Médicas

Panel de ingreso de incidente, se muestra el siguiente formulario que el auxiliar tiene que llenar con el incidente que presente el estudiante

Aplicación VELA

Registrar Incidente Médico

Nombre del Estudiante

Tipo de Incidente

¿Ya desayunó?

¿Tomó medicamentos?

¿Desea retirarse?

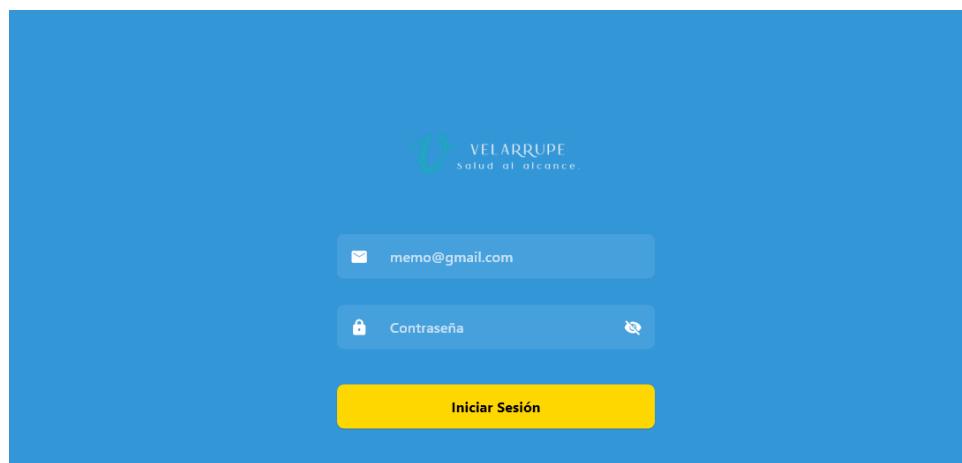
Otro (si aplica)

Registrar Incidente

Home Register Incident Fichas Médicas

Aplicación web

Inicio de sesión para ingresar credenciales de administrador del sistema web



Panel principal del administrador donde puede ver un resumen de la gestión del sistema

A screenshot of the Velarrupe administrator dashboard. The left sidebar contains navigation links: Dashboard (selected), Estudiantes, Fichas Médicas, Usuarios, Vínculos, Reporte de incidentes, and Incidentes. The main content area is titled 'Panel de Administrador' and includes sections for 'Nuestra Misión' (Mission Statement) and 'Nuestra Visión' (Vision Statement). It also features three summary cards: 'ESTUDIANTES' (4 students), 'FICHAS MÉDICAS' (2 medical forms), and 'INCIDENTES ACTIVOS' (2 active incidents). Below these are two tables: 'Últimas Atenciones' (Recent Attendances) listing Daniel Vasquez, Pablo Palacios, Raul Sorto, and Luis Guillermo; and 'Medicamentos' (Medications) listing Acetaminofén, Clorfenamina, Ibuprofeno, and Anaflat.

Panel de lista de estudiantes donde el administrador los puede registrar

The screenshot shows a web-based application interface titled "Velarrupe". On the left, a dark sidebar menu lists several options: Dashboard, Estudiantes (selected), Fichas Médicas, Usuarios, Vínculos, Reporte de incidentes, and Incidentes. The main content area is titled "Lista de Estudiantes". It features a search bar at the top with the placeholder "Buscar por nombre, apellido o carnet...". Below the search bar is a button labeled "Nuevo Estudiante". A table header row is visible with columns for ID, Nombre, Apellido, Carnet, and Acciones. The table body is currently empty.

Panel de gestión donde se pueden crear fichas médicas a los estudiantes

The screenshot shows a web-based application interface titled "Velarrupe". The sidebar menu is identical to the previous screenshot, with "Fichas Médicas" now selected. The main content area is titled "Gestión de Fichas Médicas". It includes a search bar and a button labeled "Crear Nueva Ficha Médica". A table header row is visible with columns for ID, Estudiante, Alergias, Enfermedades Crónicas, Medicamentos Crónicos, and Acciones. The table body is currently empty.

Panel de gestión de usuarios, donde se puede mostrar su rol dentro del sistema

Gestión de Usuarios

Buscar por nombre, email o rol..

Usuarios Registrados

ID	Nombre	Email	Rol	Acciones
----	--------	-------	-----	----------

Panel de gestión de vínculos para poder enlazar un estudiante con su responsable familiar

Gestión de Vínculos

Buscar por nombre de estudiante o responsable familiar...

Vínculos Registrados

ID	Estudiante	Responsable	Acciones
----	------------	-------------	----------

Panel para mostrar la lista de incidentes que presentan los estudiantes

This screenshot shows the 'Listado de Incidentes' (Incident List) page. The left sidebar has a dark blue background with white icons and text. It includes links for Dashboard, Estudiantes, Fichas Médicas, Usuarios, Vinculos, Reporte de incidentes (which is highlighted in blue), and Incidencias. The main area has a light gray header with the title 'Lista de Incidentes'. Below it is a search bar with placeholder text 'Buscar por nombre, incidente o estado..'. A button 'Reportar Nuevo Incidente' is located at the top right of the main content area. The main content area features a table header with columns: ID, Estudiante, Incidente, Fecha y Hora, Estado de Retiro, Medicamentos Tomados, Última Comida, and Acciones.

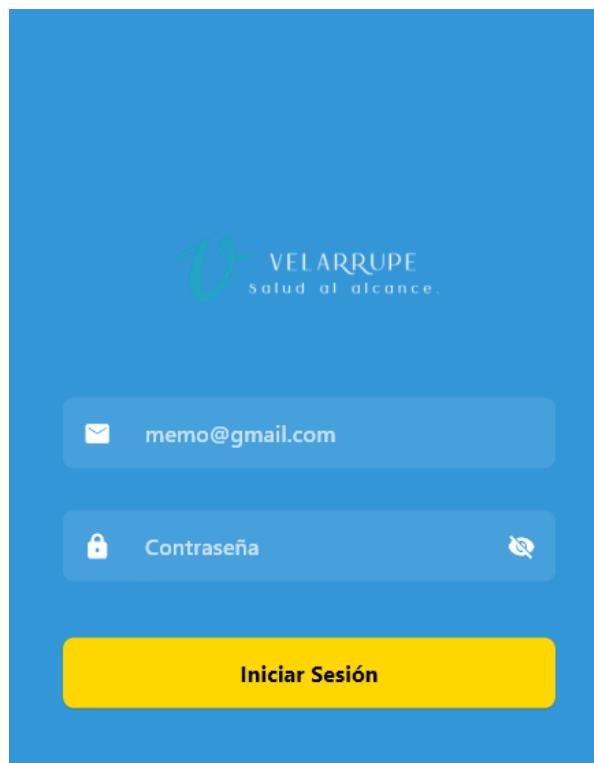
Panel para poder gestionar los incidentes que los estudiantes estén presentando

This screenshot shows the 'Gestión de Incidentes' (Incident Management) page. The left sidebar is identical to the previous one. The main area has a light gray header with the title 'Gestión de Incidentes'. Below it is a search bar with placeholder text 'Buscar por nombre..'. A button 'Crear Nuevo Incidente' is located at the top right of the main content area. The main content area features a table header with columns: ID, Nombre, and Acciones.

Aplicación móvil

Ya que la aplicación móvil contiene las funciones de las demás aplicaciones, sólo se mostrarán las partes principales del mismo

Panel de inicio de sesión, para que puedan ingresarse con credenciales de auxiliar, administrador y responsable familiar



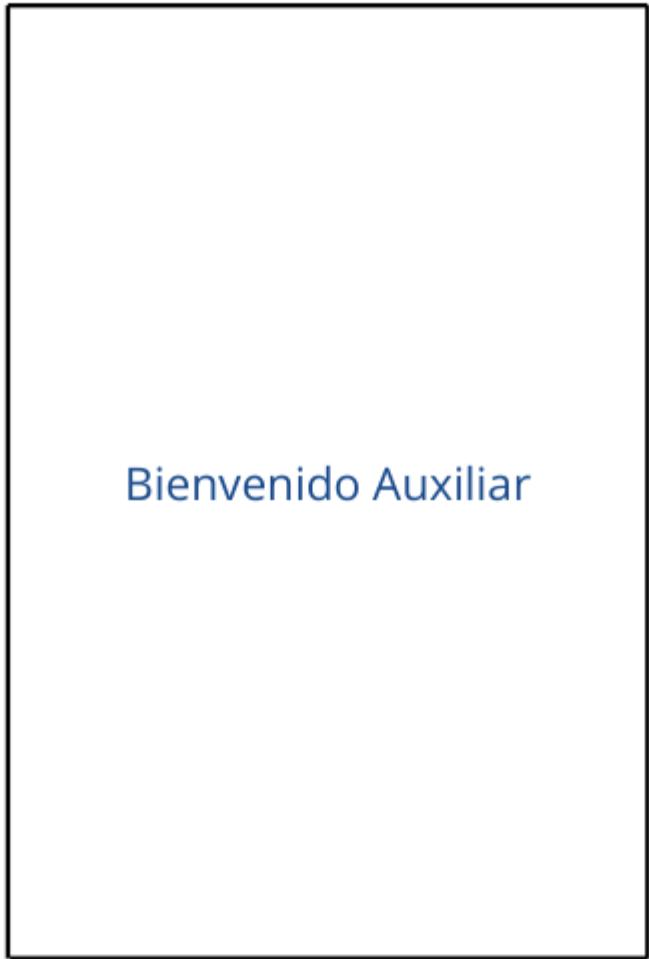
Panel de administrador, en el cual se puede realizar las diferentes acciones que este mismo puede tener

← AdminScreen



Vista Auxiliar

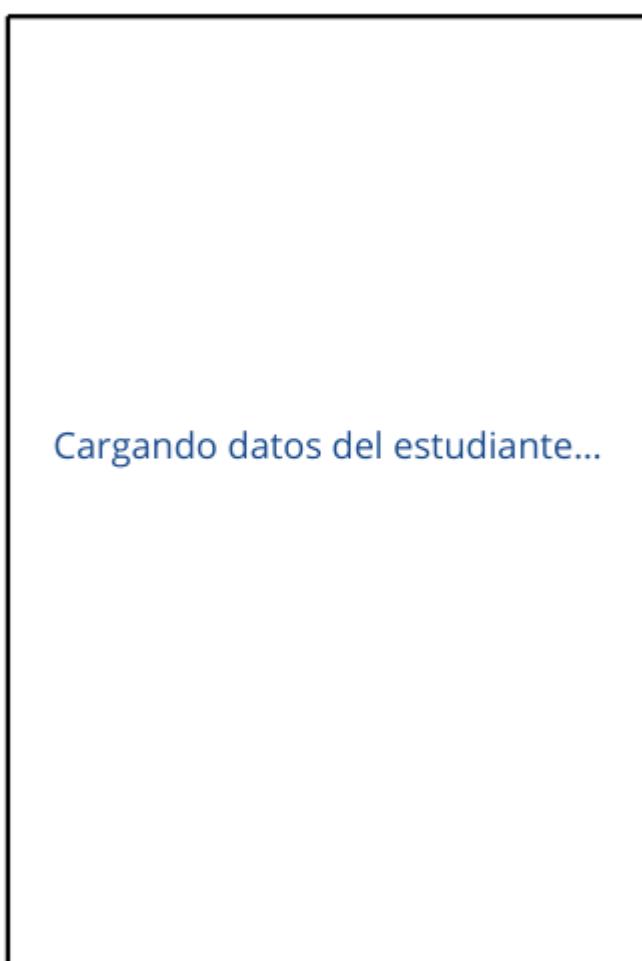
Aca se muestra los detalles que puede realizar el auxiliar, poder registrar un incidente, o ver el perfil médico



Bienvenido Auxiliar

Vista Responsable familiar

Ver los incidentes de su hijo, como responsable familiar para que así esté informado en tiempo real del estado de su hijo



3.1.1 Códigos explicativos

Aplicación de escritorio

Para la aplicación de escritorio se utilizó Flet, el cual es un framework de código abierto para facilitar el desarrollo de aplicaciones web, a continuación los códigos con detalles sobre cómo lo conforman en el sistema

api_service.py

```
import requests

API_URL = "http://localhost:8081/api/"

def obtener_usuarios():
    response = requests.get(f"{API_URL}usuarios")
    if response.status_code == 200:
        return response.json()
    return []

def obtener_fichas_medicas():
    response = requests.get(f"{API_URL}fichasmedicas")
    if response.status_code == 200:
        return response.json()
    return []

def obtener_estudiantes():
    response = requests.get(f"{API_URL}estudiantes")
    if response.status_code == 200:
        return response.json()
    return []

def obtener_estudiante(id_estudiante):
    response = requests.get(f"{API_URL}estudiantes/{id_estudiante}")
    if response.status_code == 200:
        return response.json()
    return None

def obtener_responsable(id_responsable):
    response = requests.get(f"{API_URL}usuarios/{id_responsable}")
    if response.status_code == 200:
        return response.json()
    return None

def obtener_vinculos():
    response = requests.get(f"{API_URL}vinculos")
    if response.status_code == 200:
        return response.json()
    return []
```

```

def obtener_tipos_incidentes():
    response = requests.get(f"{API_URL}incidentes")
    if response.status_code == 200:
        return response.json()
    return []

def login(email, contraseña):
    usuarios = obtener_usuarios()
    usuario = next((user for user in usuarios if user["email"] == email), None)
    if usuario and usuario["password"] == contraseña:
        return usuario
    return None

def registrar_incidente(incidente):
    response = requests.post(
        f"{API_URL}incidente-reportes",
        json=incidente, # Enviar el cuerpo como JSON
        headers={"Content-Type": "application/json"}
    )
    if response.status_code == 200:
        return response.json()
    else:
        print(f"Error al registrar incidente: {response.status_code} - {response.text}")
    return None

def obtener_medicamentos_permitidos():
    response = requests.get(f"{API_URL}medicamentos-permitidos")
    if response.status_code == 200:
        return response.json()
    return []

def obtener_medicamentos():
    response = requests.get(f"{API_URL}medicamentos")
    if response.status_code == 200:
        return response.json()
    return []

```

Este archivo define funciones para interactuar con la API REST del sistema. Entre ellas, se incluyen las siguientes: obtener usuarios, fichas médicas, estudiantes, y detalles específicos de estudiantes y responsables mediante solicitudes GET. También valida el inicio de sesión a través de email y contraseña, permite registrar incidentes médicos mediante una solicitud POST, y obtiene listas de medicamentos permitidos y en general para los estudiantes.

home_view.py:

```
import flet as ft
from views.registrar_incidentes_view import mostrar_formulario_incidente
from views.fichas_medicas_view import mostrar_fichas_medicas
from services.api_service import obtener_medicamentos_permitidos, obtener_medicamentos

def vista_principal(page: ft.Page, mostrar_login):
    #paleta de colores
    colors = {
        "primary": "#1A4B8C",          # Azul marino
        "secondary": "#E8EEF6",         # Color blanco con tono gris muy suaves
        "accent": "#2E7D32",           # Verde para acentos
        "background": "#F5F7FA",        # Gris azulado muy claro para fondo
        "text": "#2C3E50",             # Azul oscuro para texto
        "error": "#D32F2F",            # Rojo para errores
        "success": "#388E3C",          # Verde para mensaje de que se realizó correctamente
        "warning": "#F57C00",           # Naranja para las advertencias
        "surface": "#FFFFFF",          # Blanco para superficies de tarjetas
        "border": "#CBD5E1",            # Gris azulado para bordes
    }

    page bgcolor = colors["background"]
    page.padding = 0

    def navegar(selected_index):
        page.controls.clear()
        if selected_index == 0:
            cargar_inicio()
        elif selected_index == 1:
            mostrar_formulario_incidente(page)
        elif selected_index == 2:
            medicamentos_rel = obtener_medicamentos_permitidos()
            medicamentos = obtener_medicamentos()
            mostrar_fichas_medicas(page, medicamentos_rel, medicamentos)
        page.update()

    def cerrar_sesion(e):
        page.appbar = None
        page.navigation_bar = None
        page.controls.clear()
        mostrar_login(page)

    appBar = ft.AppBar(
        leading=ft.Icon(ft.icons.MENU),
        leading_width=40,
        title=ft.Text("Aplicación VELA", size=24, weight="bold", color=colors["surface"]),
        center_title=False,
        bgcolor=colors["primary"],
        actions=[
            ft.IconButton(
                icon=ft.icons.LOGOUT,
                tooltip="Cerrar sesión",
                on_click=cerrar_sesion,
                icon_color=colors["surface"]
            )
        ],
    )

    navbar = ft.NavigationBar(
        destinations=[
            ft.NavigationDestination(icon=ft.icons.HOME, label="Inicio"),
            ft.NavigationDestination(icon=ft.icons.ADD_CHART, label="Registrar Incidente"),
            ft.NavigationDestination(icon=ft.icons.FOLDER, label="Fichas Médicas"),
        ],
        on_change=lambda e: navegar(e.control.selected_index),
        bgcolor=colors["primary"],
    )

    def create_card(title, description, icon):
        return ft.Card(
            content=ft.Container(
                content=ft.Column(
                    [
                        ft.Icon(icon, color=colors["primary"], size=40),
                        ft.Text(title, size=20, weight=ft.FontWeight.BOLD, color=colors["primary"]),
                        ft.Text(description, size=14, color=colors["text"], text_align=ft.TextAlign.CENTER),
                    ],
                    alignment=ft.MainAxisAlignment.CENTER,
                    horizontal_alignment=ft.CrossAxisAlignment.CENTER,
                    spacing=10,
                ),
                padding=20,
                bgcolor=colors["surface"],
            ),
            elevation=4,
            width=300,
        )
    
```

```

def cargar_inicio():
    page.add(
        ft.Container(
            content=ft.Column(
                [
                    ft.Container(
                        content=ft.Text(
                            "VELA - Auxiliar de Planta",
                            size=32,
                            color=colors["primary"],
                            weight=ft.FontWeight.BOLD
                        ),
                        padding=20,
                        alignment=ft.alignment.center,
                        bgcolor=colors["secondary"]
                    ),
                    ft.Row(
                        [
                            create_card(
                                "Registrar Incidente",
                                "Reporta incidentes médicos de forma rápida y sencilla.",
                                ft.icons.ADD_ALERT
                            ),
                            create_card(
                                "Fichas Médicas",
                                "Consulta y administra las fichas médicas de los estudiantes.",
                                ft.icons.MEDICAL_SERVICES
                            )
                        ],
                        alignment=ft.MainAxisAlignment.CENTER,
                        spacing=20,
                    ),
                    ft.Container(
                        content=ft.Text(
                            "Utilice el menú inferior para navegar entre las opciones.",
                            color=colors["text"],
                            size=16,
                            weight=ft.FontWeight.NORMAL
                        ),
                        padding=20,
                        border_radius=8,
                        bgcolor=colors["secondary"]
                    ),
                ],
                alignment=ft.MainAxisAlignment.START,
                horizontal_alignment=ft.CrossAxisAlignment.CENTER,
                spacing=30,
            ),
            padding=40,
        )
    )

page.appbar = appBar
page.navigation_bar = navbar
cargar_inicio()

```

Aca se estructura la interfaz, incluyendo una barra de navegación y una barra de aplicaciones. Permite a los usuarios navegar entre secciones como el inicio, el registro de incidentes y las fichas médicas. La navegación actualiza dinámicamente el contenido según el índice seleccionado, y la opción de cerrar sesión elimina los controles actuales y redirige a la pantalla de inicio de sesión. Todo esto se organiza de manera modular, garantizando una experiencia de usuario profesional y accesible.

login_view.py:

```
import flet as ft
from services.api_service import login
from views.home_view import vista_principal

def pagina_login(page: ft.Page):
    page.bgcolor = "#3498db" # Color de fondo azul
    page.padding = 0
    page.window_width = 400
    page.window_height = 700

    email_ref = ft.Ref()
    contrasena_ref = ft.Ref()

    def toggle_password_visibility(e):
        contrasena_ref.current.password = not contrasena_ref.current.password
        page.update()

    # Logo
    logo = ft.Image(
        src="logo.png",
        width=200,
        height=100,
        fit=ft.ImageFit.CONTAIN,
    )

    def create_input_field(icon, ref, hint_text, password=False):
        return ft.Container(
            content=ft.Row(
                [
                    ft.Container(
                        content=ft.Icon(icon, color="white", size=20),
                        width=50,
                        height=50,
                        alignment=ft.alignment.center,
                        bgcolor="#4aa3df",
                    ),
                    ft.Container(
                        content=ft.TextField(
                            ref=ref,
                            border="none",
                            password=password,
                            hint_text=hint_text,
                            hint_style=ft.TextStyle(color="white70"),
                            text_style=ft.TextStyle(color="white"),
                            content_padding=ft.padding.only(left=10, top=12, right=10, bottom=12),
                            expand=True,
                        ),
                        expand=True,
                        height=50,
                        bgcolor="#4aa3df",
                    ),
                    ft.Container(
                        content=ft.IconButton(
                            icon=ft.icons.VISIBILITY_OFF,
                            icon_color="white",
                            icon_size=20,
                            on_click=toggle_password_visibility,
                        ) if password else None,
                        width=50,
                        height=50,
                        bgcolor="#4aa3df",
                    ) if password else ft.Container(width=0),
                ],
                spacing=0,
            ),
            border_radius=8,
            bgcolor="#4aa3df",
        )

    return ft.Column(
        [
            logo,
            create_input_field(ft.icons.EMAIL_OUTLINE, email_ref, "Email"),
            create_input_field(ft.icons.PASSWORD, contrasena_ref, "Contraseña"),
            ft.Container(
                content=ft.Column(
                    [
                        ft.Text("Olvidaste tu contraseña?"),
                        ft.Text("Crea una nueva"),
                    ],
                    alignment=ft.alignment.end,
                ),
                padding=10,
            ),
            ft.Container(
                content=ft.Column(
                    [
                        ft.FilledButton("Iniciar Sesión", on_click=login),
                        ft.FilledButton("Crear Cuenta", on_click=vista_principal),
                    ],
                    alignment=ft.alignment.end,
                ),
                padding=10,
            ),
        ],
        alignment=ft.alignment.end,
    )

```

```

# Formulario de login
login_form = ft.Column(
    [
        logo,
        ft.Container(height=20),
        create_input_field(ft.icons.EMAIL, email_ref, "memo@gmail.com"),
        ft.Container(height=10),
        create_input_field(ft.icons.LOCK, contrasena_ref, "Contraseña", password=True),
        ft.Container(height=20),
        ft.ElevatedButton(
            content=ft.Text(
                "Iniciar Sesión",
                size=16,
                weight="bold",
            ),
            style=ft.ButtonStyle(
                color="black",
                bgcolor="#ffd700",
                shape=ft.RoundedRectangleBorder(radius=8),
            ),
            width=400,
            height=50,
            on_click=lambda e: validar_login(page, email_ref, contrasena_ref),
        ),
        ft.Container(height=20),
        ft.Text(
            "¿Olvidaste tu contraseña?",
            color="white",
            size=14,
            weight="bold",
        ),
    ],
    alignment=ft.MainAxisAlignment.CENTER,
    horizontal_alignment=ft.CrossAxisAlignment.CENTER,
)

# Contenedor principal que mantiene el formulario centrado y con ancho máximo
main_container = ft.Container(
    content=login_form,
    width=400,
    padding=20,
    alignment=ft.alignment.center,
)

```

```

# Usar un Row para centrar horizontalmente el contenedor principal
centered_row = ft.Row(
    [main_container],
    alignment=ft.MainAxisAlignment.CENTER,
    expand=True,
)

# Usar un Column para centrar verticalmente el Row
centered_column = ft.Column(
    [centered_row],
    alignment=ft.MainAxisAlignment.CENTER,
    expand=True,
)

# Añadir el column centrado a la página
page.add(centered_column)

# Hacer que la página sea responsive
page.on_resize = lambda _: page.update()

def validar_login(page: ft.Page, email_ref: ft.Ref, contrasena_ref: ft.Ref):
    email = email_ref.current.value
    contrasena = contrasena_ref.current.value

    usuario = login(email, contrasena)

    if usuario:
        if usuario.get("rolId") == 2:
            page.clean() # Limpiar la pantalla de login
            vista_principal(page, pagina_login) # Pasa la función de login
        else:
            mostrar_snackbar(page, "Acceso denegado: Solo auxiliares pueden acceder.", "error")
    else:
        mostrar_snackbar(page, "Credenciales incorrectas.", "error")

def mostrar_snackbar(page: ft.Page, mensaje: str, tipo: str = "info"):
    colors = {
        "info": "#2196F3",
        "success": "#4CAF50",
        "error": "#F44336",
    }
    page.snack_bar = ft.SnackBar(
        content=ft.Text(mensaje, color="#FFFFFF"),
        bgcolor=colors.get(tipo, colors["info"]),
    )
    page.snack_bar.open = True
    page.update()

```

La pantalla de inicio de sesión muestra un formulario para ingresar correo electrónico y contraseña, con íconos en cada campo y un botón para alternar la visibilidad de la contraseña. Al hacer clic en "Iniciar Sesión", la función validar_login envía las credenciales al servicio de autenticación. Si son válidas y el usuario tiene el rol de "auxiliar", se redirige a la vista principal; de lo contrario, se muestran mensajes de error.

registrar_incidentes_view.py:

```
estudiantes = obtener_estudiantes()
tipos_incidentes = obtener_tipos_incidentes()

estudiante_ref = ft.Ref()
incidente_ref = ft.Ref()
desayuno_ref = ft.Ref()
medicamentos_ref = ft.Ref()
retiro_ref = ft.Ref()
otro_ref = ft.Ref()

opciones_estudiantes = [ft.dropdown.Option(e['idEstudiante'], e['nombre']) for e in estudiantes]
opciones_incidentes = [ft.dropdown.Option(i['id'], i['nombre']) for i in tipos_incidentes]
opciones_si_no = [ft.dropdown.Option("Sí", "Sí"), ft.dropdown.Option("No", "No")]

def crear_dropdown(label, options, ref):
    return ft.Dropdown(
        label=label,
        hint_text=label,
        options=options,
        ref=ref,
        width=300,
        color=colors["text"],
        bgcolor=colors["surface"],
        border_color=colors["primary"],
        label_style=ft.TextStyle(color=colors["primary"]),
    )

formulario = ft.Column(
    [
        ft.Container(
            content=ft.Text(
                "Registrar Incidente Médico",
                size=24,
                color=colors["primary"],
                weight=ft.FontWeight.BOLD
            ),
            bgcolor=colors["secondary"],
            padding=10,
            width=300,
            alignment=ft.alignment.center,
        ),
        crear_dropdown("Nombre del Estudiante", opciones_estudiantes, estudiante_ref),
        crear_dropdown("Tipo de Incidente", opciones_incidentes, incidente_ref),
        crear_dropdown("¿Ya desayunó?", opciones_si_no, desayuno_ref),
        crear_dropdown("¿Tomó medicamentos?", opciones_si_no, medicamentos_ref),
        crear_dropdown("¿Desea retirarse?", opciones_si_no, retiro_ref),
        ft.TextField(
            label="Otro (si aplica)",
            ref=otro_ref,
            width=300,
            color=colors["text"],
            bgcolor=colors["surface"],
            border_color=colors["primary"],
            label_style=ft.TextStyle(color=colors["primary"]),
        ),
        ft.ElevatedButton(
            "Registrar Incidente",
            on_click=lambda _: registrar_incidente(
                page, estudiante_ref, incidente_ref, desayuno_ref,
                medicamentos_ref, retiro_ref, otro_ref
            ),
            style=ft.ButtonStyle(
                bgcolor=colors["accent"],
                color=colors["surface"],
            ),
            width=300,
        ),
    ],
    alignment=ft.MainAxisAlignment.CENTER,
    horizontal_alignment=ft.CrossAxisAlignment.CENTER,
    spacing=10,
)
```

```

        contenido_principal = ft.Container(
            content=formulario,
            alignment=ft.Alignment.center,
            expand=True,
        )
        page.add(contenido_principal)

def registrar_incidente(page, estudiante_ref, incidente_ref, desayuno_ref,
                       medicamentos_ref, retiro_ref, otro_ref):
    id_estudiante = estudiante_ref.current.value
    id_incidente = incidente_ref.current.value
    desayuno = desayuno_ref.current.value
    medicamentos = medicamentos_ref.current.value
    retiro = retiro_ref.current.value
    otro = otro_ref.current.value

    if not all([id_estudiante, id_incidente, desayuno, medicamentos, retiro]):
        mostrar_snackbar(page, "Por favor, complete todos los campos requeridos.", "error")
        return

    incidente = {
        "idEstudiante": id_estudiante,
        "idIncidente": id_incidente,
        "ultimaComida": desayuno,
        "medicamentosTomados": medicamentos,
        "idEstadoRetiro": 2 if retiro == "Si" else 1,
        "otro": otro,
        "fechaHora": datetime.now().isoformat()
    }

    resultado = enviar_incidente(incidente)

    if resultado:
        mostrar_snackbar(page, "Incidente registrado exitosamente", "success")
        limpiar_formulario(estudiante_ref, incidente_ref, desayuno_ref,
                           medicamentos_ref, retiro_ref, otro_ref)
    else:
        mostrar_snackbar(page, "Error al registrar el incidente", "error")

def limpiar_formulario(*refs):
    for ref in refs:
        if isinstance(ref.current, ft.Dropdown):
            ref.current.value = None
        elif isinstance(ref.current, ft.TextField):
            ref.current.value = ""

def mostrar_snackbar(page: ft.Page, mensaje: str, tipo: str = "info"):
    colors = {
        "primary": "#1A4BBC",
        "success": "#388E3C",
        "error": "#D32F2F",
        "warning": "#F57C00",
        "surface": "#FFFFFF",
    }

    page.snack_bar = ft.SnackBar(
        content=ft.Text(mensaje, color=colors["surface"]),
        bgcolor=colors[tipo] if tipo in colors else "primary",
    )
    page.snack_bar.open = True
    page.update()

```

Este archivo gestiona la interfaz de usuario y la lógica para registrar incidentes médicos en una aplicación desarrollada con Flet. La vista incluye formularios con menús desplegables para seleccionar al estudiante y el tipo de incidente, además de campos adicionales como si el estudiante ha desayunado o tomado medicamentos. La función principal, registrar_incidente, valida los campos, envía los datos a un servicio API y muestra mensajes de éxito o error. Tras un registro exitoso, el formulario se limpia automáticamente.

fichas_medicas_view.py:

```
def obtener_nombres_medicamentos(id_estudiante, medicamentos_rel, medicamentos):
    """Obtiene los nombres de los medicamentos permitidos para un estudiante dado."""
    ids_medicamentos = [
        rel['idMedicamento'] for rel in medicamentos_rel if rel['idEstudiante'] == id_estudiante
    ]
    nombres = [
        med['nombre'] for med in medicamentos if med['id'] in ids_medicamentos
    ]
    return ", ".join(nombres) if nombres else "N/A"

def mostrar_fichas_medicas(page: ft.Page, medicamentos_rel, medicamentos):
    colors = [
        "primary": "#1A4B8C",           # Azul marino
        "secondary": "#E8EEF6",         # Color blanco con tono gris muy suaves
        "accent": "#2E7D32",           # Verde para acentos
        "background": "#F5F7FA",        # Gris azulado muy claro para fondo
        "text": "#2C3E50",             # Azul oscuro para texto
        "error": "#D32F2F",             # Rojo para errores
        "success": "#F388E3C",          # Verde para mensaje de que se realizó correctamente
        "warning": "#F57C00",            # Naranja para las advertencias
        "surface": "#FFFFFF",           # Blanco para superficies de tarjetas
        "border": "#C8D5E1",             # Gris azulado para bordes
    ]

    page.theme_mode = "light"
    page.padding = 20
    page bgcolor = colors["background"]

    fichas = obtener_fichas_medicas()
    estudiantes = obtener_estudiantes()
    vinculos = obtener_vinculos()

    search_ref = ft.Ref()
    fichas_container = ft.Column(scroll="auto", spacing=20, horizontal_alignment="center")

    def mostrar_fichas(filtradas):
        fichas_container.controls.clear()

        fila = []
        for i, ficha in enumerate(filtradas):
            estudiante = next((e for e in estudiantes if e['idEstudiante'] == ficha['idEstudiante']), None)
            responsable = None

            if estudiante:
                responsable_id = next(
                    (v['responsableId'] for v in vinculos if v['estudianteId'] == estudiante['idEstudiante']), None
                )
                responsable = obtener_responsable(responsable_id) if responsable_id else None
                meds_text = obtener_nombres_medicamentos(estudiante['idEstudiante'], medicamentos_rel, medicamentos)

                ficha_card = ft.Card(
                    content=ft.Container(
                        content=ft.Column(
                            [
                                ft.Container(
                                    content=ft.Text(
                                        f"ID: {ficha['id_fichamedica']}",
                                        weight="bold",
                                        size=20,
                                        color=colors["surface"]
                                    ),
                                    bgcolor=colors["primary"],
                                    padding=10,
                                    border_radius=ft.border_radius.only(top_left=10, top_right=10),
                                ),
                                ft.Container(
                                    content=ft.Column(
                                        [
                                            ft.Text(
                                                f'{estudiante['nombre']} {estudiante['apellido']}',
                                                size=18,
                                                weight="bold",
                                                color=colors["text"]
                                            ),
                                            ft.Text(
                                                f'Carnet: {estudiante['carnet']}',
                                                size=16,
                                                color=colors["text"]
                                            ),
                                            ft.Divider(height=1, color=colors["border"]),
                                            ft.Text(
                                                "Información Médica",
                                                weight="bold",
                                                size=16,
                                                color=colors["primary"]
                                            ),
                                            ft.Text(
                                                f'Alergias: {ficha['alergias']}',
                                                size=14,
                                                color=colors["text"]
                                            ),
                                            ft.Text(
                                                f'Enfermedades Crónicas: {ficha['enfermedadesCronicas']}',
                                                size=14,
                                                color=colors["text"]
                                            )
                                        ]
                                    )
                                )
                            ]
                        )
                    )
                )
                fila.append(ficha_card)
            if len(fila) == 10:
                break
        page.add(fichas_container)
```

```

def filtrar_fichas(e):
    criterio = search_ref.current.value.lower()
    fichas_filtradas = []

    for ficha in fichas:
        estudiante = next((e for e in estudiantes if e['idEstudiante'] == ficha['idEstudiante']), None)
        if estudiante:
            nombre_completo = f'{estudiante['nombre']} {estudiante['apellido']}'.lower()
            if criterio in nombre_completo or criterio in estudiante['carnet'].lower():
                fichas_filtradas.append(ficha)

    mostrar_fichas(fichas_filtradas)

# Mostrar todas las fichas al inicio
mostrar_fichas(fichas)

# Crear la interfaz con diseño mejorado y centrado
page.add(
    ft.Column(
        [
            # Contenedor del título centrado con fondo
            ft.Container(
                content=ft.Text(
                    "Buscar Fichas Médicas",
                    size=32,
                    weight="bold",
                    color=colors["primary"],
                    text_align="center",
                ),
                alignment=ft.alignment.center,
                margin=ft.margin.only(bottom=20),
            ),
            # Barra de búsqueda mejorada
            ft.Container(
                content=ft.TextField(
                    ref=search_ref,
                    label="Buscar por nombre o carnet",
                    prefix_icon=ft.icons.SEARCH,
                    on_change=filtrar_fichas,
                    border_radius=10,
                    filled=True,
                    bgcolor=colors["surface"],
                    border_color=colors["primary"],
                    cursor_color=colors["primary"],
                    focused_border_color=colors["primary"],
                    focused_bgcolor=colors["surface"],
                    hint_text="Ej: Juan Pérez o 12345",
                    color=colors["text"],
                    label_style=ft.TextStyle(color=colors["primary"]),
                ),
                padding=ft.padding.symmetric(horizontal=20),
                width=600,
            ),
            # Contenedor de fichas
            fichas_container
        ],
        horizontal_alignment="center",
        spacing=20,
    )
)

```

La vista muestra las fichas médicas de los estudiantes con opciones de búsqueda. La función obtener_nombres_medicamentos filtra los medicamentos permitidos para un estudiante específico. La función mostrar_fichas_medicinas carga la interfaz de las fichas médicas, mostrando los detalles de cada estudiante, como sus enfermedades, alergias y contactos de emergencia. Los datos se presentan en tarjetas estilizadas con una paleta de colores profesional. Además, la función filtrar_fichas permite buscar fichas por nombre o número de carnet, actualizando la vista en tiempo real.

Aplicación de móvil

Para el desarrollo de la aplicación móvil, se utilizó la tecnología React Native, que permite crear componentes contenedores. Estos componentes funcionan como imágenes y cuentan con funcionalidades que permiten realizar diversas acciones dentro de un único contenedor.

A continuación las funciones principales del sistema y cómo funcionan dentro de react

LoginScreen.tsx

```
8 const LoginScreen = () => {
9   const [email, setEmail] = useState('');
10  const [password, setPassword] = useState('');
11  const [showPassword, setShowPassword] = useState(false);
12  const navigation = useNavigation();
13
14  const handleLogin = async () => {
15    try {
16      const response = await axios.post('http://192.168.1.60:8080/api/usuarios/login', { email, password });
17
18      if (response.status === 200) {
19        const { roleId, nombre, email, telefono } = response.data;
20
21        // Aquí puedes almacenar el token o cualquier otro dato que necesites
22        switch (roleId) {
23          case 1: // Admin
24            navigation.navigate('AdminScreen');
25            break;
26          case 2: // Auxiliar
27            navigation.navigate('AuxiliarScreen');
28            break;
29          case 3: // Responsable familiar
30            navigation.navigate('ResponsableFamiliarScreen', {
31              nombre,
32              email,
33              telefono
34            });
35            break;
36          default:
37            console.log('Rol no reconocido');
38        }
39      } else {
40        console.log('Error: Credenciales incorrectas');
41      }
42    } catch (error) {
43      console.log('Error en la autenticación:', error.response || error);
44    }
45  };
46}
```

useState Hooks

Se utilizan para manejar el estado del correo electrónico (email), la contraseña (password) y si la contraseña está visible o no (showPassword). Estos estados se actualizan a medida que el usuario ingresa sus credenciales.

handleLogin (Función)

Es la función encargada de manejar el proceso de autenticación del usuario. Realiza una solicitud POST a un servidor (en este caso, el backend en <http://192.168.1.60:8080/api/usuarios/login>) enviando el email y la password.

Si el servidor devuelve una respuesta exitosa (código de estado 200), la aplicación verifica el rol del usuario (rolId) y redirige a la pantalla correspondiente según su rol:

- AdminScreen para administradores.
- AuxiliarScreen para auxiliares.
- ResponsableFamiliarScreen para responsables familiares, con información adicional como nombre, email y teléfono.

KeyboardAvoidingView

Se encarga de ajustar automáticamente la interfaz cuando aparece el teclado, especialmente en dispositivos iOS y Android. Evita que los campos de entrada queden ocultos debajo del teclado.

LinearGradient

Aplica un degradado de color sobre el fondo. Mejora la estética de la pantalla y da un efecto visual agradable que va desde un azul oscuro hasta un azul claro.

Inputs de Usuario

Dos campos de entrada permiten al usuario escribir su correo electrónico y contraseña.

Correo Electrónico: Es un campo tipo TextInput con icono de correo (de Ionicons) que permite la entrada de texto de tipo correo electrónico.

Contraseña: Es otro campo TextInput con un ícono de candado, y se muestra de manera oculta por defecto. Un botón de "ojito" permite alternar la visibilidad de la contraseña.

Botón de "Iniciar Sesión":

AdminScreen.tsx

```
9  const AdminButton = ({ title, icon, onPress, index }) => [
10    <Animatable.View
11      animation="fadeInUp"
12      delay={index * 100}
13      style={styles.buttonWrapper}
14    >
15      <TouchableOpacity style={styles.button} onPress={onPress}>
16        <LinearGradient
17          colors={['#4A90E2', '#5A9DEF']}
18          start={{x: 0, y: 0}}
19          end={{x: 1, y: 1}}
20          style={styles.buttonGradient}
21        >
22          <Feather name={icon} size={28} color="#FFFFFF" />
23          <Text style={styles.buttonText}>{title}</Text>
24        </LinearGradient>
25      </TouchableOpacity>
26    </Animatable.View>
27  ];
28
29  const AdminScreen = ({ navigation }) => {
30    const buttons = [
31      { title: "Agregar Estudiante", icon: "user-plus", screen: "AddStudentScreen" },
32      { title: "Ver Estudiantes", icon: "users", screen: "ViewStudentsScreen" },
33      { title: "Agregar Usuario", icon: "user", screen: "AddUserScreen" },
34      { title: "Vincular Responsable", icon: "link", screen: "VincularResponsableScreen" },
35      { title: "Agregar Ficha Médica", icon: "file-plus", screen: "AddMedicalRecordScreen" },
36      { title: "Ver Fichas Médicas", icon: "file-text", screen: "ViewMedicalRecordsScreen" },
37      { title: "Ver Usuarios", icon: "user-check", screen: "ViewUsersScreen" },
38      { title: "Reportar Incidente", icon: "alert-circle", screen: "ReportIncidentScreen" },
39      { title: "Medicamentos Permitidos", icon: "plus-circle", screen: "MedicamentosPermitidosScreen" },
40      { title: "Incidentes Pendientes", icon: "clock", screen: "ViewPendingIncidentsScreen" },
41    ];

```

Sobre todo en esta vista son las redirecciones de los para las diversa funciones que el administrador tiene sobre el sistema.

Los botones en esta pantalla representan varias acciones que puede realizar un administrador, tales como:

- Agregar Estudiante: Navega a la pantalla para agregar un nuevo estudiante.
- Ver Estudiantes: Muestra una lista de estudiantes.
- Agregar Usuario: Lleva a la vista donde se puede agregar un nuevo usuario.
- Vincular Responsable: Permite vincular a un responsable familiar con un estudiante.
- Agregar Ficha Médica: Lleva a una pantalla para registrar información médica de un estudiante.
- Ver Fichas Médicas: Muestra las fichas médicas existentes.
- Ver Usuarios: Lleva a la vista que permite ver la lista de usuarios.
- Reportar Incidente: Permite registrar incidentes.
- Medicamentos Permitidos: Lleva a la vista para gestionar medicamentos permitidos.
- Incidentes Pendientes: Muestra los incidentes que están pendientes de resolución.

AddStudentScreen.tsx

```
5  const AddStudentScreen = ({ navigation }: any) => {
6    const [nombre, setNombre] = useState('');
7    const [apellido, setApellido] = useState('');
8    const [carnet, setCarnet] = useState('');
9    const [fechaNacimiento, setFechaNacimiento] = useState('');
10
11   const validateCarnet = (carnet: string) => {
12     // Expresión regular para 2 letras seguidas de 6 números
13     const regex = /^[A-Za-z]{2}\d{6}$/;
14     return regex.test(carnet);
15   };
16
17   const handleAddStudent = async () => {
18     if (!validateCarnet(carnet)) {
19       Alert.alert('Error', 'El carnet debe tener 2 letras seguidas de 6 números.');
20       return;
21     }
22
23     // Convertir la fecha a formato ISO antes de enviarla
24     const formattedFechaNacimiento = new Date(fechaNacimiento).toISOString().split('T')[0];
25
26     try {
27       const response = await axios.post('http://192.168.1.60:8080/api/estudiantes/add-student', {
28         nombre,
29         apellido,
30         carnet,
31         fechaNacimiento: formattedFechaNacimiento, // Enviar la fecha formateada
32       });
33       // Mostrar mensaje personalizado con el nombre del estudiante
34       Alert.alert('Éxito', `Has agregado al estudiante ${nombre}`);
35       navigation.goBack(); // Regresa a la pantalla anterior después de agregar
36     } catch (error) {
37       console.log('Error al agregar estudiante:', error);
38       Alert.alert('Error', 'No se pudo agregar el estudiante');
39     }
40   };
}
```

1- El componente utiliza el hook useState para manejar los datos del formulario. Los valores ingresados por el usuario se almacenan en el estado local:

- nombre: Almacena el nombre del estudiante.
- apellido: Almacena el apellido del estudiante.
- carnet: Almacena el carnet del estudiante. Tiene validación especial.
- fechaNacimiento: Almacena la fecha de nacimiento del estudiante.

Validación del Carnet

La función validateCarnet utiliza una expresión regular para asegurarse de que el carnet siga el formato específico de 2 letras seguidas de 6 números.

Función handleAddStudent

Este es el método clave que se ejecuta cuando el usuario presiona el botón "Agregar Estudiante":
Luego, intenta enviar los datos a un servidor con la librería axios mediante una solicitud POST al endpoint `http://192.168.1.60:8080/api/estudiantes/add-studentr`.
Convierte la fecha de nacimiento en el formato ISO adecuado (usado comúnmente en API) antes de enviarla al servidor.

AddUserScreen.tsx

```
16  const rolMapping = {
17    admin: 1,
18    auxiliar: 2,
19    responsable_familiar: 3,
20  };
21
22  const handleAddUser = async () => {
23    if (!nombre || !email || !password || !telefono) [
24      setSnackbarMessage('Todos los campos son obligatorios');
25      setVisible(true);
26      return;
27    ]
28
29    try {
30      await axios.post('http://192.168.1.60:8080/api/usuarios', {
31        nombre,
32        email,
33        password,
34        telefono, // Añadimos el teléfono al payload
35        roleId: rolMapping[roleId], // Enviamos el ID del rol
36      });
37      setSnackbarMessage('Usuario agregado exitosamente');
38      setVisible(true);
39      setTimeout(() => navigation.navigate('AdminScreen'), 2000);
40    } catch (error) {
41      console.error('Error al agregar usuario:', error);
42      setSnackbarMessage('No se pudo agregar el usuario');
43      setVisible(true);
44    }
45  };

```

1- Uso de useState para el manejo de estado

Se utilizan múltiples hooks de useState para manejar los valores de los campos del formulario y controlar el estado del componente:

- nombre: Nombre del usuario.
- email: Correo electrónico del usuario.
- password: Contraseña del usuario.
- telefono: Teléfono del usuario.
- roleId: Rol seleccionado para el usuario (admin, auxiliar, responsable familiar).
- visible: Controla la visibilidad del Snackbar, que muestra mensajes de confirmación o error.
- snackackbarMessage: Almacena el mensaje que se mostrará en el Snackbar.

2- Función handleAddUser

Esta es la función que se ejecuta cuando el usuario presiona el botón "Agregar Usuario". Aquí se realiza la validación de los campos obligatorios, se envían los datos al servidor, y se maneja la respuesta:

Validación: Si alguno de los campos está vacío, se muestra un Snackbar con un mensaje de error.

Envía los datos: Si todos los campos son válidos, envía una solicitud POST al servidor usando axios para agregar el usuario

MedicamentosPermitidos.tsx

```
6  const MedicamentosPermitidosScreen = () => [
7    const [students, setStudents] = useState([]);
8    const [filteredStudents, setFilteredStudents] = useState([]);
9    const [searchText, setSearchText] = useState('');
10   const [medicamentos, setMedicamentos] = useState([]);
11   const [selectedStudent, setSelectedStudent] = useState('');
12   const [selectedMedicamentos, setSelectedMedicamentos] = useState([]);

13
14 // obtenemos a los estudiantes y medicamentos desde la base de datos
15 useEffect(() => {
16   const fetchData = async () => {
17     try {
18       const studentsResponse = await axios.get('http://192.168.1.60:8080/api/estudiantes');
19       setStudents(studentsResponse.data);
20       setFilteredStudents(studentsResponse.data);
21
22       const medicamentosResponse = await axios.get('http://192.168.1.60:8080/api/medicamentos');
23       setMedicamentos(medicamentosResponse.data.map(medicamento => {
24         id: medicamento.id,
25         name: medicamento.nombre
26       }));
27     } catch (error) {
28       console.error('Error al obtener datos:', error);
29     }
30   };
31
32   fetchData();
33 }, []);

34 // manejamos la búsqueda de estudiantes
35 const handleSearch = (text) => {
36   setSearchText(text);
37   const filtered = students.filter(student =>
38     student.nombre.toLowerCase().includes(text.toLowerCase())
39   );
40   setFilteredStudents(filtered);
41 };
42 
```

1-Estados Iniciales y Hooks

Se utilizan varios estados (useState) para manejar los datos que cambian a lo largo del ciclo de vida del componente:

- students: Lista de estudiantes obtenida desde la API.

- filteredStudents: Lista filtrada de estudiantes basada en el término de búsqueda.
- searchText: Texto ingresado en la barra de búsqueda.
- medicamentos: Lista de medicamentos permitidos obtenidos desde la API.
- selectedStudent: ID del estudiante seleccionado.
- selectedMedicamentos: Lista de IDs de los medicamentos seleccionados.

El hook useEffect es utilizado para obtener la lista de estudiantes y medicamentos desde la API cuando el componente se monta. El código dentro de useEffect hace llamadas a dos endpoints:

<http://192.168.1.60:8080/api/estudiantes>: Para obtener la lista de estudiantes.

<http://192.168.1.60:8080/api/medicamentos>: Para obtener la lista de medicamentos permitidos.

```

44 // manejamos el envío del reporte
45 const handleSubmit = async () => {
46   if (!selectedStudent || selectedMedicamentos.length === 0) {
47     Alert.alert('Error', 'Por favor selecciona un estudiante y al menos un medicamento.');
48     return;
49   }
50
51   // creamos un array de reportes solo con los nuevos medicamentos
52   const reportes = selectedMedicamentos.map(medicamentoId => ({
53     idEstudiante: selectedStudent,
54     idMedicamento: medicamentoId
55   }));
56
57   try {
58     // hacemos multiples consultas a las base de datos con los medicamentos permitidos por
59     await Promise.all(reportes.map(reporte =>
60       axios.post('http://192.168.1.60:8080/api/medicamentos-permitidos', reporte)
61     ));
62     Alert.alert('Éxito', 'Los medicamentos permitidos han sido guardados correctamente.');
63     // Limpiar los campos
64     setSelectedStudent('');
65     setSelectedMedicamentos([]);
66     setSearchText('');
67   } catch (error) {
68     console.error('Error al guardar medicamentos permitidos:', error);
69
70     if (error.response && error.response.status === 400) {
71       Alert.alert('Error', 'Este estudiante ya tiene estos medicamentos permitidos.');
72     } else {
73       Alert.alert('Error', 'Hubo un problema al guardar los medicamentos.');
74     }
75   }
76 }

```

1- Envío de Medicamentos Permitidos

La función handleSubmit maneja el envío de la lista de medicamentos permitidos seleccionados para el estudiante. El flujo general es el siguiente:

- Primero, se valida que se haya seleccionado un estudiante y al menos un medicamento.
- Luego, se crea un array llamado reportes, que contiene los objetos con las IDs del estudiante y los medicamentos seleccionados.
- Se utiliza Promise.all para hacer múltiples solicitudes POST al endpoint <http://192.168.1.60:8080/api/medicamentos-permitidos>, donde cada solicitud envía un par de ID de estudiante y medicamento.
- Si todo se guarda correctamente, se muestra una alerta de éxito, y los campos seleccionados se restablecen.

En caso de error, se manejan las respuestas erróneas con alertas apropiadas, incluyendo un manejo específico para errores 400 (Bad Request), que indica que el estudiante ya tiene los medicamentos seleccionados.

Aplicación web

Para este sistema se resumió y se usó de base los controladores ya que es donde se realizan la mayoría de acciones

EstudianteController.java

```
@Controller
@RequestMapping("/admin/estudiantes")
public class EstudianteController {
    @Autowired
    private EstudianteService estudianteService;

    @Autowired
    private FichaMedicaService fichaMedicaService;

    @GetMapping
    public String listarEstudiantes(Model model) {
        model.addAttribute("estudiantes", estudianteService.findAll());
        return "admin/estudiantes/lista";
    }

    @GetMapping("/crear")
    public String mostrarFormularioCrear(Model model) {
        model.addAttribute("estudiante", new Estudiante());
        return "admin/estudiantes/form";
    }

    @PostMapping("/crear")
    public String crearEstudiante(@ModelAttribute Estudiante estudiante, RedirectAttributes redirectAttributes) {
        try {
            estudianteService.save(estudiante);
            redirectAttributes.addFlashAttribute("success", "Estudiante creado exitosamente");
        } catch (Exception e) {
            redirectAttributes.addFlashAttribute("error", "Error al crear el estudiante: " + e.getMessage());
        }
        return "redirect:/admin/estudiantes";
    }

    @GetMapping("/editar/{id}")
    public String mostrarFormularioEditar(@PathVariable Integer id, Model model) {
        model.addAttribute("estudiante", estudianteService.findById(id));
        return "admin/estudiantes/form";
    }

    @PostMapping("/editar/{id}")
    public String actualizarEstudiante(@PathVariable Integer id, @ModelAttribute Estudiante estudiante,
                                       RedirectAttributes redirectAttributes) {
        try {
            estudiante.setIdEstudiante(id);
            estudianteService.update(id, estudiante);
            redirectAttributes.addFlashAttribute("success", "Estudiante actualizado exitosamente");
        } catch (Exception e) {
            redirectAttributes.addFlashAttribute("error", "Error al actualizar el estudiante: " + e.getMessage());
        }
        return "redirect:/admin/estudiantes";
    }

    @GetMapping("/eliminar/{id}")
    public String eliminarEstudiante(@PathVariable Integer id, RedirectAttributes redirectAttributes) {
        try {
            // Verificar si existe una ficha médica para el estudiante
            if (fichaMedicaService.existeFichaMedicaPorEstudiante(id)) {
                redirectAttributes.addFlashAttribute("error",
                        "Este estudiante tiene una ficha médica asociada. Por favor, elimínela primero.");
            } else {
                // Proceder a eliminar el estudiante
                estudianteService.deleteById(id);
                redirectAttributes.addFlashAttribute("success", "Estudiante eliminado exitosamente");
            }
        } catch (Exception e) {
            redirectAttributes.addFlashAttribute("error", "Para eliminar a este estudiante primero borre su ficha")
        }
        return "redirect:/admin/estudiantes";
    }
}
```

Gestiona la administración de estudiantes, permitiendo listar estudiantes, crear, editar y eliminar registros

FichaMedicaController.java

```
@Controller
@RequestMapping("/admin/fichas-medicas")
public class FichaMedicaController {

    @Autowired
    private FichaMedicaService fichaMedicaService;

    @Autowired
    private EstudianteService estudianteService;

    @GetMapping
    public String listarFichasMedicas(Model model) {
        List<FichaMedica> fichasMedicas = fichaMedicaService.findAll();
        for (FichaMedica ficha : fichasMedicas) {
            String nombreEstudiante = estudianteService.findById(ficha.getIdEstudiante()).getNombre();
            ficha.setNombreEstudiante(nombreEstudiante);
        }
        model.addAttribute("fichasMedicas", fichasMedicas);
        return "admin/fichas-medicas/listar";
    }

    @GetMapping("/crear")
    public String mostrarFormularioCrear(Model model) {
        model.addAttribute("fichaMedica", new FichaMedica());
        model.addAttribute("estudiantes", estudianteService.findAll());
        return "admin/fichas-medicas/crear";
    }

    @PostMapping("/crear")
    public String crearFichaMedica(@ModelAttribute FichaMedica fichaMedica) {
        fichaMedicaService.save(fichaMedica);
        return "redirect:/admin/fichas-medicas";
    }

    @GetMapping("/editar/{id}")
    public String mostrarFormularioEditar(@PathVariable Integer id, Model model) {
        FichaMedica fichaMedica = fichaMedicaService.findById(id);
        String nombreEstudiante = estudianteService.findById(fichaMedica.getIdEstudiante()).getNombre();
        fichaMedica.setNombreEstudiante(nombreEstudiante);
        model.addAttribute("fichaMedica", fichaMedica);
        model.addAttribute("estudiantes", estudianteService.findAll());
        return "admin/fichas-medicas/editar";
    }

    @PostMapping("/editar/{id}")
    public String actualizarFichaMedica(@PathVariable Integer id, @ModelAttribute FichaMedica fichaMedica) {
        fichaMedica.setId_fichamedica(id);
        fichaMedicaService.update(id, fichaMedica);
        return "redirect:/admin/fichas-medicas";
    }

    @GetMapping("/eliminar/{id}")
    public String eliminarFichaMedica(@PathVariable Integer id) {
        fichaMedicaService.deleteById(id);
        return "redirect:/admin/fichas-medicas";
    }
}
```

Gestiona las fichas médicas, realiza las acciones de listar, crear, editar y eliminar registros.

Proporciona rutas para mostrar todas las fichas médicas con el nombre del estudiante asociado, crear nuevas fichas médicas, editar fichas existentes y eliminar registros. Las acciones redirigen a la lista de fichas médicas y aseguran que la interfaz se mantenga actualizada.

IncidenteController.java

```
@Controller
@RequestMapping("/admin/incidentes")
public class IncidenteController {

    @Autowired
    private IncidenteService incidenteService;

    @GetMapping
    public String listarIncidentes(Model model) {
        model.addAttribute("incidentes", incidenteService.findAll());
        return "admin/incidentes/listar";
    }

    @GetMapping("/crear")
    public String mostrarFormularioCrear(Model model) {
        model.addAttribute("incidente", new Incidente());
        return "admin/incidentes/crear";
    }

    @PostMapping("/crear")
    public String crearIncidente(@ModelAttribute Incidente incidente) {
        incidenteService.save(incidente);
        return "redirect:/admin/incidentes";
    }

    @GetMapping("/editar/{id}")
    public String mostrarFormularioEditar(@PathVariable Integer id, Model model) {
        model.addAttribute("incidente", incidenteService.findById(id));
        return "admin/incidentes/editar";
    }

    @PostMapping("/editar/{id}")
    public String actualizarIncidente(@PathVariable Integer id, @ModelAttribute Incidente incidente) {
        incidente.setId(id);
        incidenteService.update(id, incidente);
        return "redirect:/admin/incidentes";
    }

    @GetMapping("/eliminar/{id}")
    public String eliminarIncidente(@PathVariable Integer id) {
        incidenteService.deleteById(id);
        return "redirect:/admin/incidentes";
    }
}
```

Gestiona los incidentes, permitiendo listar, crear, editar y eliminar registros de incidentes. Proporciona rutas para mostrar todos los incidentes en una lista, mostrar un formulario para crear un nuevo incidente, editar un incidente existente y eliminar un registro. Las acciones redirigen a la lista de incidentes para mantener actualizada la interfaz de usuario.

IncidenteReporteController.java

```
@Controller
@RequestMapping("/admin/incidentes-reportes")
public class IncidenteReporteController {
    @Autowired
    private IncidenteReporteService incidenteReporteService;

    @Autowired
    private EstudianteService estudianteService;

    @Autowired
    private IncidenteService incidenteService;

    @GetMapping
    public String listarIncidentesReportes(Model model) {
        model.addAttribute("incidentesReportes", incidenteReporteService.findAllWithNames());
        return "admin/incidentes-reportes/listar";
    }

    @GetMapping("/{id}")
    public String verIncidenteReporte(@PathVariable Integer id, Model model) {
        IncidenteReporte incidenteReporte = incidenteReporteService.findByIdWithNames(id);
        if (incidenteReporte == null) {
            return "redirect:/admin/incidentes-reportes";
        }
        model.addAttribute("incidenteReporte", incidenteReporte);
        return "admin/incidentes-reportes/ver";
    }

    @GetMapping("/crear")
    public String mostrarFormularioCrear(Model model) {
        model.addAttribute("incidenteReporte", new IncidenteReporte());
        model.addAttribute("estudiantes", estudianteService.findAll());
        model.addAttribute("incidentes", incidenteService.findAll());
        return "admin/incidentes-reportes/crear";
    }

    @PostMapping("/crear")
    public String crearIncidenteReporte(@ModelAttribute IncidenteReporte incidenteReporte) {
        incidenteReporteService.save(incidenteReporte);
        return "redirect:/admin/incidentes-reportes";
    }

    @GetMapping("/editar/{id}")
    public String mostrarFormularioEditar(@PathVariable Integer id, Model model) {
        IncidenteReporte incidenteReporte = incidenteReporteService.findByIdWithNames(id);
        if (incidenteReporte == null) {
            return "redirect:/admin/incidentes-reportes";
        }
        model.addAttribute("incidenteReporte", incidenteReporte);
        model.addAttribute("estudiantes", estudianteService.findAll());
        model.addAttribute("incidentes", incidenteService.findAll());
        return "admin/incidentes-reportes/editar";
    }
```

```

    @PostMapping("/editar/{id}")
    public String actualizarIncidenteReporte(@PathVariable Integer id, @ModelAttribute IncidenteReporte incidenteReporte) {
        incidenteReporte.setId(id);
        incidenteReporteService.update(id, incidenteReporte);
        return "redirect:/admin/incidentes-reportes";
    }

    @PostMapping("/{id}/aprobar")
    public String aprobarRetiro(@PathVariable Integer id) {
        incidenteReporteService.aprobarRetiro(id);
        return "redirect:/admin/incidentes-reportes";
    }

    @PostMapping("/{id}/rechazar")
    public String rechazarRetiro(@PathVariable Integer id) {
        incidenteReporteService.rechazarRetiro(id);
        return "redirect:/admin/incidentes-reportes"; // Redirige a la lista después de rechazar
    }

    @GetMapping("/eliminar/{id}")
    public String eliminarIncidenteReporte(@PathVariable Integer id) {
        incidenteReporteService.deleteById(id);
        return "redirect:/admin/incidentes-reportes";
    }
}

```

Gestiona los reportes de incidentes en una aplicación web, permitiendo listar, crear, ver, editar y eliminar reportes. Proporciona rutas para listar todos los reportes con información adicional, ver detalles de un reporte específico, crear nuevos reportes incluyendo listas de estudiantes y tipos de incidentes, y editar reportes existentes. También permite aprobar o rechazar retiros relacionados con reportes y eliminar reportes, asegurando que la interfaz de usuario se mantenga actualizada tras cada acción.

LoginController.java

```
@Controller
public class LoginController {

    @Autowired
    private UsuarioService usuarioService;

    @GetMapping("/login")
    public String loginForm() {
        return "login";
    }

    @PostMapping("/login")
    public String login(@RequestParam String username,
                        @RequestParam String password,
                        HttpSession session,
                        Model model) {
        if (usuarioService.authenticate(username, password)) {
            session.setAttribute("user", username);
            return "redirect:/admin/dashboard";
        } else {
            model.addAttribute("error", "Invalid username or password");
            return "login";
        }
    }

    @GetMapping("/logout")
    public String logout(HttpSession session, HttpServletResponse response) {
        session.invalidate();

        response.setHeader("Cache-Control", "no-cache, no-store, must-revalidate");
        response.setHeader("Pragma", "no-cache");
        response.setDateHeader("Expires", 0);

        return "redirect:/login";
    }

    @GetMapping("/admin/dashboard")
    public String dashboard(HttpSession session, HttpServletResponse response) {
        // Verificar si el usuario está autenticado
        if (session.getAttribute("user") == null) {
            return "redirect:/login"; // Redirige al login si no está autenticado
        }

        response.setHeader("Cache-Control", "no-cache, no-store, must-revalidate");
        response.setHeader("Pragma", "no-cache");
        response.setDateHeader("Expires", 0);

        return "admin/dashboard"; // Muestra el dashboard si está autenticado
    }

    // Método para verificar autenticación en todas las URLs
    @GetMapping("/admin/**") // Aplica a todas las URL bajo el contexto de admin
    public String checkAuth(HttpSession session) {
        if (session.getAttribute("user") == null) {
            return "redirect:/login"; // Redirige al login si no está autenticado
        }
        return null; // Permite que la solicitud continúe si el usuario está autenticado
    }
}
```

Maneja autenticación de usuarios. Valida las credenciales mediante UsuarioService. Si son correctas, redirige al dashboard; de lo contrario, muestra un error. Al cerrar sesión, invalida la sesión y evita el almacenamiento en caché. También verifica la autenticación en rutas de administración, redirigiendo a los no autenticados al login.

MedicamentoController.java

```
@Controller
@RequestMapping("/admin/medicamentos")
public class MedicamentoController {

    @Autowired
    private MedicamentoService medicamentoService;

    @GetMapping
    public String listarMedicamentos(Model model) {
        model.addAttribute("medicamentos", medicamentoService.findAll());
        return "admin/medicamentos/listar";
    }

    @GetMapping("/crear")
    public String mostrarFormularioCrear(Model model) {
        model.addAttribute("medicamento", new Medicamento());
        return "admin/medicamentos/crear";
    }

    @PostMapping("/crear")
    public String crearMedicamento(@ModelAttribute Medicamento medicamento) {
        medicamentoService.save(medicamento);
        return "redirect:/admin/medicamentos";
    }

    @GetMapping("/editar/{id}")
    public String mostrarFormularioEditar(@PathVariable Integer id, Model model) {
        model.addAttribute("medicamento", medicamentoService.findById(id));
        return "admin/medicamentos/editar";
    }

    @PostMapping("/editar/{id}")
    public String actualizarMedicamento(@PathVariable Integer id, @ModelAttribute Medicamento medicamento) {
        medicamento.setId(id);
        medicamentoService.update(id, medicamento);
        return "redirect:/admin/medicamentos";
    }

    @GetMapping("/eliminar/{id}")
    public String eliminarMedicamento(@PathVariable Integer id) {
        medicamentoService.deleteById(id);
        return "redirect:/admin/medicamentos";
    }
}
```

El controlador gestiona medicamentos en una aplicación Spring MVC. Permite listar, crear, editar y eliminar medicamentos usando MedicamentoService. Proporciona métodos para mostrar formularios y manejar las solicitudes de creación y actualización, redirigiendo a la lista de medicamentos tras cada acción.

MedicamentoPermitidoController.java

```
@Controller
@RequestMapping("/admin/medicamentos-permitidos")
public class MedicamentoPermitidoController {

    @Autowired
    private MedicamentoPermitidoService medicamentoPermitidoService;

    @Autowired
    private EstudianteService estudianteService;

    @Autowired
    private MedicamentoService medicamentoService;

    @GetMapping
    public String listarMedicamentosPermitidos(Model model) {
        model.addAttribute("medicamentosPermitidos", medicamentoPermitidoService.findAll());
        return "admin/medicamentos-permitidos/listar";
    }

    @GetMapping("/crear")
    public String mostrarFormularioCrear(Model model) {
        model.addAttribute("medicamentoPermitido", new MedicamentoPermitido());
        model.addAttribute("estudiantes", estudianteService.findAll());
        model.addAttribute("medicamentos", medicamentoService.findAll());
        return "admin/medicamentos-permitidos/crear";
    }

    @PostMapping("/crear")
    public String crearMedicamentoPermitido(@ModelAttribute MedicamentoPermitido medicamentoPermitido) {
        medicamentoPermitidoService.save(medicamentoPermitido);
        return "redirect:/admin/medicamentos-permitidos";
    }

    @GetMapping("/editar/{id}")
    public String mostrarFormularioEditar(@PathVariable Integer id, Model model) {
        model.addAttribute("medicamentoPermitido", medicamentoPermitidoService.findById(id));
        model.addAttribute("estudiantes", estudianteService.findAll());
        model.addAttribute("medicamentos", medicamentoService.findAll());
        return "admin/medicamentos-permitidos/editar";
    }

    @PostMapping("/editar/{id}")
    public String actualizarMedicamentoPermitido(@PathVariable Integer id, @ModelAttribute MedicamentoPermitido medicamentoPermitido) {
        medicamentoPermitido.setId(id);
        medicamentoPermitidoService.update(id, medicamentoPermitido);
        return "redirect:/admin/medicamentos-permitidos";
    }

    @GetMapping("/eliminar/{id}")
    public String eliminarMedicamentoPermitido(@PathVariable Integer id) {
        medicamentoPermitidoService.deleteById(id);
        return "redirect:/admin/medicamentos-permitidos";
    }
}
```

El controlador gestiona los medicamentos permitidos. Utiliza MedicamentoPermitidoService para listar, crear, editar y eliminar medicamentos permitidos. Proporciona métodos para mostrar formularios de creación y edición, además de redirigir a la lista de medicamentos permitidos tras cada operación. Incluye datos de estudiantes y medicamentos en los formularios.

RetiroController.java

```
@Controller
@RequestMapping("/admin/retiros")
public class RetiroController {

    @Autowired
    private RetiroService retiroService;

    @GetMapping
    public String listarRetiros(Model model) {
        model.addAttribute("retiros", retiroService.findAll());
        return "admin/retiros/listar";
    }

    @GetMapping("/crear")
    public String mostrarFormularioCrear(Model model) {
        model.addAttribute("retiro", new Retiro());
        return "admin/retiros/crear";
    }

    @PostMapping("/crear")
    public String crearRetiro(@ModelAttribute Retiro retiro) {
        retiroService.save(retiro);
        return "redirect:/admin/retiros";
    }

    @GetMapping("/editar/{id}")
    public String mostrarFormularioEditar(@PathVariable Integer id, Model model) {
        model.addAttribute("retiro", retiroService.findById(id));
        return "admin/retiros/editar";
    }

    @PostMapping("/editar/{id}")
    public String actualizarRetiro(@PathVariable Integer id, @ModelAttribute Retiro retiro) {
        retiro.setId(id);
        retiroService.update(id, retiro);
        return "redirect:/admin/retiros";
    }

    @GetMapping("/eliminar/{id}")
    public String eliminarRetiro(@PathVariable Integer id) {
        retiroService.deleteById(id);
        return "redirect:/admin/retiros";
    }
}
```

El controlador gestiona los retiros. Utiliza RetiroService para realizar operaciones sobre los retiros, incluyendo listar, crear, editar y eliminar. Proporciona métodos para mostrar formularios para crear y editar retiros, y redirige a la lista de retiros después de cada operación.

RolController.java

```
@Controller
@RequestMapping("/admin/roles")
public class RolController {

    @Autowired
    private RolService rolService;

    @GetMapping
    public String listarRoles(Model model) {
        model.addAttribute("roles", rolService.findAll());
        return "admin/roles/listar";
    }

    @GetMapping("/crear")
    public String mostrarFormularioCrear(Model model) {
        model.addAttribute("rol", new Rol());
        return "admin/roles/crear";
    }

    @PostMapping("/crear")
    public String crearRol(@ModelAttribute Rol rol) {
        rolService.save(rol);
        return "redirect:/admin/roles";
    }

    @GetMapping("/editar/{id}")
    public String mostrarFormularioEditar(@PathVariable Integer id, Model model) {
        model.addAttribute("rol", rolService.findById(id));
        return "admin/roles/editar";
    }

    @PostMapping("/editar/{id}")
    public String actualizarRol(@PathVariable Integer id, @ModelAttribute Rol rol) {
        rol.setId(id);
        rolService.update(id, rol);
        return "redirect:/admin/roles";
    }

    @GetMapping("/eliminar/{id}")
    public String eliminarRol(@PathVariable Integer id) {
        rolService.deleteById(id);
        return "redirect:/admin/roles";
    }
}
```

El controlador gestiona los roles, utilizando RolService para realizar operaciones CRUD sobre los roles. Los métodos permiten listar roles, mostrar formularios para crear y editar roles, y eliminar roles.

Después de cada operación, redirige a la lista de roles.

UsuarioController.java

```
@Controller
@RequestMapping("/admin/usuarios")
public class UsuarioController {
    @Autowired
    private UsuarioService usuarioService;

    @Autowired
    private RolService rolService;

    @GetMapping
    public String listarUsuarios(Model model) {
        List<Usuario> usuarios = usuarioService.findAll();
        List<Rol> roles = rolService.findAll();

        // Crear un mapa de roleId a nombre del rol
        Map<Integer, String> rolNombres = roles.stream()
            .collect(Collectors.toMap(Rol::getId, Rol::getNombre));

        // Asignar el nombre del rol a cada usuario
        usuarios.forEach(usuario ->
            usuario.setNombreRol(rolNombres.getOrDefault(usuario.getRoleId(), "No asignado"))
        );

        model.addAttribute("usuarios", usuarios);
        return "admin/usuarios/listar";
    }

    @GetMapping("/crear")
    public String mostrarFormularioCrear(Model model) {
        model.addAttribute("usuario", new Usuario());
        model.addAttribute("roles", rolService.findAll());
        return "admin/usuarios/crear";
    }

    @PostMapping("/crear")
    public String crearUsuario(
        @ModelAttribute Usuario usuario,
        RedirectAttributes redirectAttributes) {
        try {
            usuarioService.save(usuario);
            redirectAttributes.addFlashAttribute("mensaje", "Usuario creado exitosamente");
            return "redirect:/admin/usuarios";
        } catch (Exception e) {
            redirectAttributes.addFlashAttribute("error", "Error al crear el usuario: " + e.getMessage());
            return "redirect:/admin/usuarios/crear";
        }
    }

    @GetMapping("/editar/{id}")
    public String mostrarFormularioEditar(@PathVariable Integer id, Model model) {
        try {
            Usuario usuario = usuarioService.findById(id)
                .orElseThrow(() -> new IllegalArgumentException("Usuario no válido con id: " + id));

            model.addAttribute("usuario", usuario);
            model.addAttribute("roles", rolService.findAll());
            return "admin/usuarios/editar";
        } catch (Exception e) {
            model.addAttribute("error", "Error al cargar el usuario: " + e.getMessage());
            return "redirect:/admin/usuarios";
        }
    }
}
```

```

    @PostMapping("/editar/{id}")
    public String actualizarUsuario(
        @PathVariable Integer id,
        @ModelAttribute Usuario usuario,
        RedirectAttributes redirectAttributes) {
        try {
            // Asegurar que el ID del path variable se establezca en el usuario
            usuario.setId(id);

            // Si el password viene vacío, mantener el password actual
            if (usuario.getPassword() == null || usuario.getPassword().trim().isEmpty()) {
                Usuario usuarioExistente = usuarioService.findById(id)
                    .orElseThrow(() -> new IllegalArgumentException("Usuario no encontrado"));
                usuario.setPassword(usuarioExistente.getPassword());
            }

            usuarioService.update(id, usuario);
            redirectAttributes.addFlashAttribute("mensaje", "Usuario actualizado exitosamente");
            return "redirect:/admin/usuarios";
        } catch (Exception e) {
            redirectAttributes.addFlashAttribute("error", "Error al actualizar el usuario: " + e.getMessage());
            return "redirect:/admin/usuarios/editar/" + id;
        }
    }

    @GetMapping("/eliminar/{id}")
    public String eliminarUsuario(
        @PathVariable Integer id,
        RedirectAttributes redirectAttributes) {
        try {
            usuarioService.deleteById(id);
            redirectAttributes.addFlashAttribute("mensaje", "Usuario eliminado exitosamente");
            return "redirect:/admin/usuarios";
        } catch (Exception e) {
            redirectAttributes.addFlashAttribute("error", "Error al eliminar el usuario: " + e.getMessage());
            return "redirect:/admin/usuarios";
        }
    }
}

```

El controlador UsuarioController gestiona las operaciones para los usuarios en una aplicación Spring MVC, utilizando UsuarioService y RolService. Permite listar usuarios con sus roles, crear nuevos usuarios, editar usuarios existentes y eliminar usuarios. Cada operación redirige a la lista de usuarios o muestra el formulario correspondiente, mostrando mensajes de éxito o error según sea necesario.

VinculoController.java

```
@Controller
@RequestMapping("/admin/vinculos")
public class VinculoController {
    @Autowired
    private VinculoService vinculoService;
    @Autowired
    private EstudianteService estudianteService;
    @Autowired
    private UsuarioService usuarioService;

    @GetMapping
    public String listarVinculos(Model model) {
        try {
            List<Vinculo> vinculos = vinculoService.findAllWithDetails();
            model.addAttribute("vinculos", vinculos);
        } catch (Exception e) {
            System.err.println("Error loading vinculos: " + e.getMessage());
            model.addAttribute("error", "Ha ocurrido un error al cargar los vínculos. Por favor, inténtelo de nuevo más tarde.");
        }
        return "admin/vinculos/listar";
    }

    @GetMapping("/crear")
    public String mostrarFormularioCrear(Model model) {
        model.addAttribute("vinculo", new Vinculo());
        model.addAttribute("estudiantes", estudianteService.findAll());
        model.addAttribute("responsables", usuarioService.findAllResponsables());
        return "admin/vinculos/crear";
    }

    @PostMapping("/crear")
    public String crearVinculo(@ModelAttribute Vinculo vinculo, Model model) {
        try {
            System.out.println("Vinculo a crear: " + vinculo); // Imprime el objeto para depuración
            vinculoService.save(vinculo);
            return "redirect:/admin/vinculos";
        } catch (Exception e) {
            e.printStackTrace(); // Imprime el stack trace completo
            model.addAttribute("error", "Error al crear el vínculo: " + e.getMessage());
            model.addAttribute("estudiantes", estudianteService.findAll());
            model.addAttribute("responsables", usuarioService.findAllResponsables());
            return "admin/vinculos/crear";
        }
    }

    @GetMapping("/editar/{id}")
    public String mostrarFormularioEditar(@PathVariable Integer id, Model model) {
        try {
            model.addAttribute("vinculo", vinculoService.findByIdWithDetails(id));
            model.addAttribute("estudiantes", estudianteService.findAll());
            model.addAttribute("responsables", usuarioService.findAllResponsables());
            return "admin/vinculos/editar";
        } catch (Exception e) {
            System.err.println("Error loading vinculo for edit: " + e.getMessage());
            model.addAttribute("error", "Ha ocurrido un error al cargar el vínculo para editar. Por favor, inténtelo de nuevo.");
            return "redirect:/admin/vinculos";
        }
    }
}
```

```

    @PostMapping("/editar/{id}")
    public String actualizarVinculo(@PathVariable Integer id, @ModelAttribute Vinculo vinculo, Model model) {
        try {
            vinculo.setId(id);
            vinculoService.update(id, vinculo);
            return "redirect:/admin/vinculos";
        } catch (Exception e) {
            System.err.println("Error updating vinculo: " + e.getMessage());
            model.addAttribute("error", "Ha ocurrido un error al actualizar el vínculo. Por favor, inténtelo de nuevo.");
            return "admin/vinculos/editar";
        }
    }

    @GetMapping("/eliminar/{id}")
    public String eliminarVinculo(@PathVariable Integer id, Model model) {
        try {
            vinculoService.deleteById(id);
            return "redirect:/admin/vinculos";
        } catch (Exception e) {
            System.err.println("Error deleting vinculo: " + e.getMessage());
            model.addAttribute("error", "Ha ocurrido un error al eliminar el vínculo. Por favor, inténtelo de nuevo.");
            return "redirect:/admin/vinculos";
        }
    }
}

```

El VinculoController gestiona las operaciones CRUD para los vínculos. Permite listar vínculos con detalles, crear nuevos vínculos, editar y eliminar existentes. En cada operación, maneja errores y proporciona mensajes adecuados, redirigiendo al usuario a la lista de vínculos o mostrando el formulario correspondiente. Además, al crear o editar un vínculo, carga listas de estudiantes y responsables para completar la información.

3.2 Plan de Implementación

3.2.1 Método de Implementación

La implementación del sistema informático para la vigilancia estudiantil y logística de atención se llevará a cabo siguiendo un enfoque estructurado que asegure la efectividad y la aceptación del sistema por parte de los usuarios finales.

Despliegue

El sistema se implementará en los edificios de la institución, específicamente en el área auxiliar, con el objetivo de optimizar la gestión de incidentes de los estudiantes. La implementación se llevará a cabo en varias fases:

Capacitación

Poder capacitar al personal del colegio, auxiliar y administradores para que sean capaces de poder manejar el sistema de acuerdo a lo establecido previamente en este documento, con las instrucciones necesarias para que puedan hacer uso del mismo.

Recursos

Recurso de Hardware: El uso de equipos que brinda la institución donde se podrá usar y operar el sistema.

Recurso Software: El sistema previo para poder atender las situaciones que los estudiantes presenten durante la jornada lectiva.

3.2.2 Configuración e Instalación

Este proceso abarca la configuración necesaria de la instalación del sistema

Se deben realizar las siguientes acciones:

Configuración de la Base de Datos y API

Configuración de la Base de Datos: De acuerdo con las instrucciones proporcionadas en los manuales, es fundamental establecer la base de datos que almacenará toda la información. En este contexto, la API es responsable de manejar la lógica necesaria para la creación y gestión de dichos datos.

Instalación de la Aplicación

Seguir todos los pasos descritos en los manuales, asegurando que el sistema esté completamente operativo para el soporte de los procesos de registro de incidentes. Cabe destacar que estos documentos se presentarán de manera separada a este informe.

Nota: Las configuraciones, la creación de la base de datos y la instalación de la aplicación se describen con mayor detalle en los manuales

- 1-Manual de Auxiliar
- 2- Manual de Administrador

3.2.3 Implementación de un servidor

La implementación de un servidor es importante para asegurar la disponibilidad y el rendimiento del sistema informático para la vigilancia estudiantil y logística de atención. A continuación, se describen las consideraciones necesarias para llevar a cabo esta implementación:

1. Selección del Hardware del Servidor

Se deberá seleccionar un servidor adecuado que cumpla con los requisitos de capacidad y rendimiento del sistema. Este servidor deberá contar con:

Procesador: Mínimo de cuatro núcleos para manejar múltiples solicitudes simultáneamente.

Memoria RAM: Al menos 16 GB para soportar la carga de trabajo del sistema.

Almacenamiento: Disco duro SSD para un acceso rápido a la base de datos y aplicaciones.

Conectividad: Puertos de red de alta velocidad para garantizar la comunicación eficiente con los dispositivos en la institución.

2. Configuración del Servidor

La configuración del servidor incluirá los siguientes pasos:

Instalación del Sistema Operativo: Recomendamos Windows para poder configurarlo debido a su estabilidad y seguridad del entorno.

Configuración de la Red: Establecimiento de direcciones IP, configuración de firewalls y ajustes de seguridad para proteger la red.

3. Pruebas de Funcionamiento

Una vez que el servidor esté configurado, se debe de llevar a cabo pruebas exhaustivas para asegurar que:

El sistema responde adecuadamente bajo diferentes cargas de trabajo.

La conectividad con los dispositivos en la institución es estable.

La seguridad del servidor está garantizada y los accesos no autorizados están bloqueados.

4. Mantenimiento y Soporte Técnico

Se establecerá un plan de mantenimiento para el servidor, que incluya:

Actualizaciones de Software: Implementación regular de actualizaciones de seguridad y mejoras de rendimiento.

Monitoreo del Rendimiento: Uso de herramientas para monitorear el rendimiento del servidor y detectar problemas antes de que afecten a los usuarios.

Soporte Técnico: Designación de un equipo de soporte que pueda resolver problemas técnicos de manera oportuna.

3.3 Requisitos mínimos

Para garantizar un rendimiento adecuado y una experiencia de usuario óptima, se establecen los siguientes requisitos mínimos para la instalación y ejecución de las aplicaciones

Aplicación de escritorio

Sistema Operativo: Windows 10 o superior

Memoria RAM: 8 GB

Espacio en disco: 2 GB para la aplicación

Procesador: Procesador intel i5-1135G7 @ 2.40GHz

Aplicación de celular

Sistema Operativo: Android Oreo 8.1

Memoria RAM: 4 GB

Espacio en disco: 2 GB

Procesador: Snapdragon 732G

Aplicación web

Sistema Operativo: Windows 10 y superior

Java: JDK 21

Memoria RAM: 8 GB

Espacio en disco: 2 GB

Procesador: Procesador intel i5-1135G7 @ 2.40GHz

3.4 Conclusiones

Al concluir el desarrollo del proyecto sistema informático de vigilancia estudiantil y logística de atención , podemos concluir que:

Los tres roles (admin, auxiliar y responsable) son esenciales para el funcionamiento de la aplicación. La correcta gestión de los permisos y la experiencia de usuario personalizada para cada uno garantiza que el sistema sea seguro, eficiente y fácil de usar.

Al combinar un sistema web en Java, una app móvil en React Native y una aplicación de escritorio en Python, se está cubriendo un amplio uso de plataformas. Asegurando que el mismo flujo de trabajo pueda ser llevado a cabo sin problemas al comunicarse entre sí.

Dado que la aplicación maneja datos médicos y perfiles de usuarios, la seguridad debe ser una prioridad máxima. Además, optimizar el rendimiento en cada plataforma garantizará que los usuarios, especialmente los auxiliares que necesitan acceder rápidamente a información crítica, puedan hacerlo sin demoras.

El uso de tecnologías diversas, como Java, React Native y Python, muestra la flexibilidad del sistema. Esto permite que el sistema pueda ser escalable en el futuro, integrando más funciones en la arquitectura principal.

3.5 Recomendaciones

- 1- La aplicación de móvil en el futuro pueden ser ejecutables, lo que permitirá al personal del colegio y responsables familiares poder instalarlos.
- 2- El sistema puede ser escalable en el futuro para poder abarcar un número de datos mayor de estudiantes.
- 3- Es necesario realizar pruebas con usuarios reales de cada rol (administrador, auxiliar y responsable) para garantizar que todas las funciones y permisos se comporten correctamente.
- 4- Aunque el sistema ya cumple con las funcionalidades establecidas, se recomienda garantizar que la estructura de la API y la base de datos sea escalable, de modo que pueda soportar futuras expansiones y mejoras.

3.6 Bibliografías

Sitios Web

- 1- *Sobre la Matriz FODA*
<https://asana.com/es/resources/swot-analysis>
- 2- *Sobre los casos de uso*
<https://www.ibm.com/docs/es/product-master/12.0.0?topic=processes-defining-use-cases>
- 3- *Sobre las historias de usuario*
<https://www.atlassian.com/es/agile/project-management/user-stories>
- 4- *Diagrama de causa y efecto*
<https://blog.hubspot.es/sales/diagrama-ishikawa>

Tesis Utilizada como Referencia

- *Sistema Informático para la administración y seguimiento de las actividades académicas ex-aulas del Plan Anual Escolar del Colegio Español Padre Arrupe - Repositorio Institucional de la Universidad de El Salvador. (s. f.).*
<https://oldri.ues.edu.sv/id/eprint/19461/>