

PROYECTO AURORA

ARANGO DÍAZ SAMUEL

FLOREZ JULIO JEISON STEVEN

UNIVERSIDAD PONTIFICIA BOLIVARIANA

ESCUELA DE INGENIERÍAS

FACULTAD DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

JUAN CARLOS MARINO MORANTES (Docente Universitario)

BUCARAMANGA, COLOMBIA

2024

ÍNDICE

1. Introducción.....	3
2. Situación Problema.....	4
2.1 Pregunta Problema.....	5
3. Justificación.....	6
4. Objetivo General.....	7
4.1 Objetivos Específicos.....	8
5. Metodología.....	9
5.1 Fases del proyecto.....	10
6. Marco tecnológico.....	12
7. Marco Conceptual.....	14
8. Marco Legal.....	16
9. Diseños.....	21
9.1 Modelo de datos.....	22
9.2 Diagramas.....	24
9.2.1 Diagrama de clase.....	25
9.2.2 Diagrama de casos de uso.....	26
9.2.3 Diagrama de secuencia.....	27
9.2.4 Diagrama de actividades.....	28
9.3 Actividades.....	29
9.4 Interfaces.....	30
9.4.1 Vistas de diseño.....	31
9.4.2 Visualización de código.....	45
10. Especificación de Requerimientos.....	90
11. Implementaciones.....	92
11.1 Tecnologías.....	93
11.2 Software.....	95
11.3 Pruebas.....	98
12. Cronograma.....	102
13. Resultados.....	103
14. Conclusiones.....	106
15. Referencias Bibliográficas.....	107

1. Introducción

El presente documento aborda el desarrollo de un sistema integral de gestión de citas médicas diseñado para optimizar los procesos operativos de un centro médico privado de alta calidad denominado Proyecto Aurora, situado en la localidad de Bucaramanga, Colombia. Este proyecto surge como respuesta a la imperiosa necesidad de implementar una plataforma de asistencia y gestión médica eficiente y especializada, dirigida específicamente al ámbito de la salud, con el objetivo de mejorar la experiencia de los usuarios y garantizar la excelencia en la atención médica brindada. Donde el sistema incluye autorización de órdenes, autorización de exámenes, atención de citas médicas con el médico de su preferencia y especialidad deseada.

Ofrecemos una visión integral que busca armonizar las expectativas de eficiencia, optimización y excelencia en la gestión de citas médicas. Este enfoque integrador tiene como objetivo principal satisfacer las necesidades tanto de los usuarios como del personal médico, garantizando una atención fluida y de alta calidad en todo el proceso de gestión de citas.

El sistema busca atender los desafíos de largas esperas y gestiones tardías de citas y exámenes en el área médica, con el propósito de crear procesos internos en el centro de salud de forma segura, rápida, mejorando la experiencia del paciente. Además, contará con el cumplimiento de los reglamentos legales de la manipulación de datos personales, como son la confidencialidad de la información de cada uno de los usuarios, satisfaciendo las necesidades del cuidado de datos sensibles que se desempeña principalmente en el ámbito a tratar.

A continuación, se detalla la situación con base a la problemática, la justificación y metodología a seguir en el desarrollo del sistema, junto con la presentación de los respectivos marcos tecnológicos, conceptuales y de requerimientos. Asimismo, se incluye un cronograma de actividades diseñado para orientar el proceso de implementación del sistema de gestión de citas médicas bajo el nombre de proyecto Aurora.

2. Situación Problema

En la localidad de Bucaramanga, se identifica la necesidad de implementar un sistema de gestión de pacientes en un centro de salud privado. Enfocándonos en dicha necesidad, es importante establecer este nuevo centro que permita a los residentes de Bucaramanga gestionar sus citas y exámenes de manera eficiente.

Uno de los problemas más comunes que se presentan en la organización interna de las EPS (Entidades Promotoras de Salud) son las dificultades para manejar adecuadamente el sistema de gestión de citas y exámenes médicos, tanto para el personal como para los pacientes al agendar, programar o cancelar citas y exámenes de distintas especialidades. Estos problemas suelen derivarse de diversos factores, como un mal funcionamiento del sistema, un diseño poco intuitivo para navegar entre las funciones y la falta de capacidad del personal del centro de salud para utilizar el sistema de manera efectiva, entre otros.

Entre los numerosos desafíos que pueden surgir, la gestión eficiente de las colas de espera para citas y exámenes, así como la autorización de órdenes dentro de las especialidades disponibles, destaca como uno de los más importantes. En algunas sedes de salud, la situación actual se caracteriza por largas horas de espera para los pacientes, tanto en la solicitud de citas como en la realización de exámenes necesarios. Esta congestión no solo genera incomodidad e insatisfacción en los usuarios, sino que también cuestiona la eficacia de un sistema de salud en el cual el servicio de atención médica no satisface las necesidades de los usuarios.

Además, la falta de un sistema de autorización de órdenes para exámenes dentro de las especialidades disponibles puede resultar en retrasos en el tratamiento y diagnóstico de los pacientes, lo cual puede tener un impacto negativo en su salud y bienestar.

Este proyecto no solo busca abordar los desafíos actuales en la atención médica, sino también establecer las bases para un modelo de atención centrado en el paciente, donde el sistema y la tecnología desempeñen un papel fundamental. La implementación exitosa de este sistema beneficiará no solo a la comunidad de Bucaramanga en general, sino que también proporcionará una atención médica satisfactoria y accesible en un entorno local.

2.1 Pregunta Problema

¿Cómo implementar un sistema de gestión de pacientes en Java para el agendamiento de citas, la administración de filas de espera y la autorización de órdenes para exámenes médicos en un centro de salud privado en Bucaramanga, asegurando un despliegue adecuado en el entorno clínico?

3. Justificación

El desarrollo del proyecto busca implementar un sistema de gestión para el apartado de citas médicas y su correspondiente despliegue en un entorno clínico en el área metropolitana de Bucaramanga, con el propósito de mejorar la eficiencia y competitividad en el sector de la salud. La automatización del proceso de toma de citas, autorización y registro de exámenes contribuirá a una expansión en el sistema, mejorando los procesos internos del centro de salud y la experiencia de los usuarios, lo que se traducirá en una mayor satisfacción del cliente. Además, el sistema será escalable para hacer frente al crecimiento de la demanda y futuras expansiones.

La automatización del proceso de agendamiento de citas y exámenes médicos permitirá una atención médica más eficiente, mejorando significativamente la experiencia del usuario/paciente. Al eliminar las largas filas y el tedioso proceso de agendamiento manual, el sistema proporcionará una forma más conveniente para que los pacientes accedan a la atención médica que necesitan.

Además, la implementación de este sistema de gestión de citas médicas aportará conocimientos y beneficios tanto al campo tecnológico como al sector de la salud en general en el área metropolitana de Bucaramanga. Este proyecto servirá como un ejemplo de cómo la tecnología puede transformar y mejorar los procesos en el ámbito de la salud, fomentando la adopción de soluciones innovadoras en otros centros médicos y clínicas de la región.

En este proyecto se contribuirá al avance del campo tecnológico y el mejoramiento del sistema de atención de citas médicas y el sector en general. También buscará elevar los estándares de calidad, eficiencia y accesibilidad en los servicios de salud, beneficiando directamente a los pacientes, al personal médico y administrativo, así como a la comunidad en su conjunto. En resumen, la implementación del sistema de gestión de citas será fundamental para el crecimiento sostenible del proyecto Aurora y la innovación en la atención médica en el área metropolitana de Bucaramanga, permitiendo un mejor uso del tiempo del personal médico y reduciendo los tiempos inactivos, lo que mejorará la productividad del centro de salud.

4. Objetivo General

El objetivo principal de este proyecto es diseñar, desarrollar e implementar un sistema de gestión de pacientes para un sistema integral de atención médica en la ciudad de Bucaramanga, con el fin de mejorar la eficiencia operativa y la calidad de la atención proporcionada. Este sistema estará basado en la metodología de espiral, que permite una adaptación continua a medida que se van identificando y resolviendo los requisitos y desafíos del proyecto.

Utilizando el lenguaje de programación Java y adoptando las mejores prácticas de desarrollo de software, el sistema se diseñará para ofrecer una interfaz intuitiva y amigable para el usuario, garantizando una experiencia fluida tanto para el personal médico como para los pacientes. Además, se emplearán archivos de texto (.txt) para la persistencia y el almacenamiento de datos, asegurando la integridad y confidencialidad de la información del paciente.

El propósito principal de este sistema es automatizar y optimizar el proceso completo de gestión de pacientes en el entorno de salud. Esto incluye, pero no se limita a, el agendamiento eficiente de citas médicas, la autorización oportuna de exámenes clínicos, el registro preciso de información del paciente y la prestación de una atención médica oportuna y de calidad por parte del personal debidamente calificado.

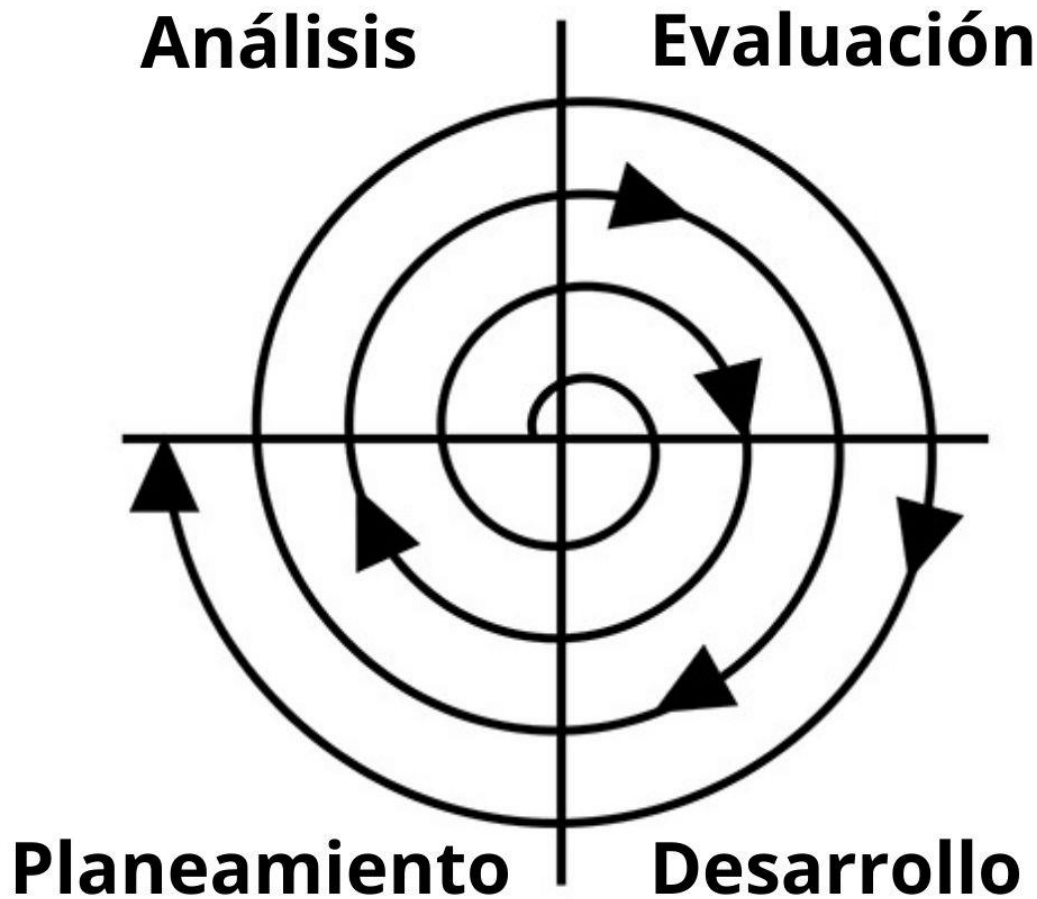
Se espera que este sistema abarque diversas especialidades médicas, permitiendo la gestión integral de pacientes con diferentes necesidades de atención médica. Se priorizará la interoperabilidad con otros sistemas de salud y la integración de estándares de seguridad y privacidad de datos para garantizar la confidencialidad y protección de la información del paciente.

4.1 Objetivos Específicos

- Definir indicadores de calidad para el proceso de agendamiento de citas, estableciendo métricas para evaluar la calidad del proceso de agendamiento que permitirá identificar áreas que se deben tratar con prioridad y a futuro mejorar con base a cómo evoluciona el sistema y garantizar una atención óptima para los pacientes.
- Diseñar una interfaz visualmente cómoda para los usuarios del sistema donde se busca ir de la mano con la funcionalidad y adaptabilidad de una forma práctica y precisa, el diseño de una interfaz que cubra todas las implicaciones que demandan contribuirá de forma positiva a la eficacia general del sistema y a su buena reputación.
- Desarrollar un prototipo funcional del sistema, centrándonos en los requerimientos que se definen por el cliente, donde la creación de un prototipo funcional permitirá validar y refinar los requisitos del sistema antes de avanzar en etapas que requieren de una atención mayor del desarrollo.
- Optimizar la gestión de citas y registros médicos mediante el desarrollo del sistema, donde la implementación de sistemas de gestión robustos como la aplicación de las estructuras de datos y la programación orientada a objetos/eventos es vital para asegurar la eficiencia y optimización en la gestión de citas y registros sin comprometer la privacidad de los pacientes.
- Evaluar el funcionamiento integral del sistema asegurando el cumplimiento de los requisitos acordados con el cliente y satisfacer las necesidades del centro de salud, esta etapa de evaluación exhaustiva busca garantizar que el sistema cumpla con los estándares acordados y sea capaz de satisfacer las necesidades específicas del centro de salud.
- Mitigar los riesgos potenciales asociados al desarrollo e implementación del sistema y eliminar algún tipo de error, este objetivo es fundamental para identificar posibles obstáculos y establecer estrategias para abordarlos antes de la entrega del producto garantizando una base sólida para el desarrollo del proyecto.
- Garantizar la claridad y precisión en la accesibilidad a los servicios, la interfaz de usuario y el funcionamiento general del sistema deben ser intuitivos y fácil de usar donde es esencial asegurar que los usuarios puedan interactuar con el sistema de manera efectiva y obtener los servicios que necesitan de manera clara, oportuna y precisa.

5. Metodología

La metodología empleada para el desarrollo del sistema es la metodología de espiral. Esta metodología se fundamenta en una planificación meticulosa del proyecto, la identificación y evaluación exhaustiva de riesgos, un enfoque iterativo de desarrollo y una evaluación continua, lo que permite minimizar eficientemente los errores. Avanzar en las tareas solo se llevará a cabo si se completa un sprint con eficacia y se cumplen las expectativas establecidas. El sistema basado en esta metodología permitirá crear un programa de Asistente de Citas Médicas eficiente, completo y con una reducción de los riesgos potenciales durante todo el proceso de creación. Se garantiza, de esta manera, que sólo se avanzará a la siguiente fase si la fase actual está completamente finalizada, asegurando así el correcto funcionamiento de la tarea en curso.



5.1 Fases del proyecto

- Análisis de Riesgos

Se identificarán los posibles riesgos asociados al desarrollo e implementación del sistema. Entre los riesgos potenciales se pueden incluir problemas de carga de datos, retrasos en la asignación de exámenes y citas, fallos en la integración de la sala de espera, suplantación de identidad, entre otros.

Se evaluará el impacto y la probabilidad de ocurrencia de cada riesgo de manera rigurosa, utilizando matrices de riesgo u otras herramientas apropiadas. Se diseñarán planes de contingencia y mitigación detallados para reducir su impacto en el proyecto.

- Evaluación y planificación

Se llevarán a cabo planificaciones y evaluaciones detalladas. En estas planificaciones se definirán funcionalidades que debe incluir y los requisitos necesarios para su correcto funcionamiento y aplicación de estructuras sólidas para garantizar una correcta modularidad y mantenimiento.

Se establecerá un cronograma para el desarrollo del sistema y se asignan los recursos necesarios para llevar a cabo el proyecto. Además, los roles y responsabilidades de los miembros del equipo encargados del desarrollo serán de forma donde contribuirán de forma colectiva para una mejora continua desde distintos puntos.

Asimismo, se analizaron minuciosamente los requisitos del sistema, centrándose en las necesidades específicas de un Asistente de Citas Médicas, y se recopiló información detallada sobre las funcionalidades requeridas.

- Desarrollo y prueba

Se seguirá un enfoque iterativo comenzando con la definición de los requisitos del usuario y la especificación de las funcionalidades clave del sistema.

Se procederá a la creación de un prototipo inicial del sistema, y se realizarán ciclos de refinamiento y mejora para garantizar la calidad del producto final.

Durante esta etapa, se implementarán funcionalidades como el Agendamiento de Citas, la autorización de exámenes específicos, el registro de usuarios, el seguimiento médico, entre otros.

- Planeamiento

Se llevarán a cabo pruebas exhaustivas y planificaciones detalladas. Todas las funcionalidades del sistema serán probadas para identificar problemas y oportunidades de mejora. Se implementarán las correcciones y mejoras necesarias para asegurar que el sistema funcione de manera óptima y cumpla con los requisitos establecidos en la fase de planificación. Además, se realizará una evaluación continua del rendimiento del sistema utilizando métricas relevantes, como el tiempo de procesamiento para agendar citas, la asignación correspondiente de profesionales al programar un examen y la eficacia de los resultados obtenidos.

6. Marco tecnológico

El presente marco tecnológico aborda el panorama actual de las tecnologías que están en juego dentro del proyecto. En un mundo donde la innovación tecnológica avanza a un ritmo vertiginoso, es fundamental comprender las herramientas, plataformas y sistemas que impactan directamente para la correcta realización de un asistente de citas médicas que cubren todos los requerimientos demandados por el cliente e ir más allá de la satisfacción de este mismo.

1. Plataforma de Desarrollo

- **IDE**, Un entorno de desarrollo integrado (IDE) es un sistema de software para el diseño de aplicaciones que combina herramientas comunes para desarrolladores en una sola interfaz gráfica de usuario (GUI). Se utiliza el IDE NetBeans para el despliegue de la aplicación. Sin embargo, cada participante en el desarrollo del código del programa.

2. Entorno de Desarrollo integrado (IDE)

- **NetBeans**, Es un entorno de desarrollo integrado, gratuito y de código abierto para el desarrollo de aplicaciones en los sistemas operativos Windows, Mac, Linux y Solaris. Desarrollaremos el proyecto y realizaremos el completo despliegue en el IDE NetBeans. Siendo así, un IDE popular y de positiva confianza para el desarrollo de aplicaciones Java que ofrece múltiples ventajas y herramientas útiles para la edición de código, depuración y pruebas.

3. Lenguaje de programación

- **Java**, Java es un tipo de lenguaje de programación y una plataforma informática, creada y comercializada por Sun Microsystems en el año 1995. Es funcional para este proyecto, ya que se constituye como un lenguaje orientado a objetos el cual está fuertemente equipado, permitiendo que los desarrolladores implicados en la creación de este sistema escriban el programa una sola vez y lo ejecuten en cualquier sistema operativo y es altamente adoptado en el desarrollo empresarial ofreciendo una plataforma confiable y escalable.

4. Control de versiones

- **Git**, Git es una herramienta que realiza el sistema de control de versiones de código de forma distribuida. Es de código abierto, con mantenimiento activo y es la herramienta de este tipo más empleada en el mundo. Utilizaremos Git como el sistema de control de versiones para rastrear y administrar los cambios en el código, creando así diferentes ramificaciones para mantener un historial de desarrollo confiable y sólido, y asegurar la fiabilidad del proyecto.

5. Diseños de interfaces

- **Figma**, Esta es una herramienta para diseñar prototipos, wireframes e interfaces. Todo aquello que posea una interfaz gráfica se puede diseñar desde Figma, ya sean páginas web, pantallas de móvil e incluso para smartwatches. Utilizaremos Figma para diseñar interfaces gráficas de usuario (GUI) del sistema. Figma permite crear prototipos interactivos y diseños de alta calidad que facilitan la comunicación visual entre el equipo de diseño y el desarrollo. Y aunque es una plataforma para el diseño más que todo implementado en aplicativos webs, se usó para el trabajo en tiempo real con las personas pertenecientes al equipo de trabajo.
- **IntelliJ IDEA**, Este es un entorno de desarrollo integrado(IDE) que ofrece características avanzadas para el desarrollo de aplicaciones Java, incluidas las aplicaciones de escritorio con JavaFX. Tiene un editor de GUI visual que te permite diseñar interfaces de usuario de forma visual y luego generar el código correspondiente. Se aplicó también la creación de distintas vistas desde otras plataformas de desarrollo para ver su capacidad de creación de diseño.
- **JavaFX**, Esta es una plataforma de software donde las funcionalidades para la creación de interfaces de usuario modernas y dinámicas en aplicaciones de escritorio. Proporciona una variedad de componentes gráficos, animaciones, efectos visuales y herramientas para el desarrollo de aplicaciones interactivas. Permitiendo diseñar y obtener el código del diseño implementado.

6. Persistencia de la información

- **Archivos de texto**, Los archivos de texto son una forma común y sencilla de almacenar la información de manera legible para los humanos y fácilmente manipulable por los dispositivos, en este caso las computadoras. Son ampliamente utilizados en una variedad de aplicaciones debido a su simplicidad y portabilidad. Los datos almacenados en archivos de texto pueden ser fácilmente editados, compartidos, almacenados de forma constante entre diferentes sistemas y aplicaciones. Además, los archivos de texto son adecuados para almacenar información estructurada o no estructurada, como registros de eventos, configuraciones de aplicaciones, datos de configuración, entre otros. Su simplicidad y práctica y versatilidad los convierten en una opción popular para la persistencia de datos informáticos.

7. Marco Conceptual

Este marco conceptual proporciona una base fundamental para comprender los elementos clave relacionados con el funcionamiento y los desafíos inherentes al proyecto para el correcto funcionamiento de este donde se pueden identificar áreas de mejora, diseñar estrategias para aumentar la eficiencia y mejorar el proceso de agendamiento de citas y asignación de exámenes médicos yendo de la mano con la satisfacción del cliente.

- El ciclo de vida del desarrollo de sistemas (SDLC) es aquel proceso que abarca el desarrollo de nuevo software desde la etapa de planificación inicial hasta la implementación y el mantenimiento a largo plazo. Es una herramienta de mapeo que ayuda a los desarrolladores de software a medida que crean un nuevo software.
- Esto yendo de la mano con la metodología de espiral que fue aplicada, permite hacer una combinación entre el modelo WaterFall y un modelo por iteraciones. Donde el proceso pasa por distintas etapas, desde la de conceptualización siguiendo el desarrollo, luego una fase de mejoras, para finalizar con el mantenimiento y así haciendo el proyecto de una forma más escalable y práctica.
- Aplicando a su vez las estructuras de datos para la creación de un proyecto mucho más modular y sólido. Implementando las ventajas como son la aplicación de las Listas enlazadas, que son aquellas que generan un vínculo entre una lista de elementos importantes de un programa. Organizando el orden de la información dentro del sistema cuando se ejecuta un programa de gestión de información, a su vez la aplicación de Listas doblemente enlazadas, pilas, colas las cuales nos permiten avanzar como retroceder. Cada nodo de la lista replicada tiene dos referencias, una al siguiente nodo y otra al anterior, además del campo de datos. Se utiliza un enlace a la derecha para avanzar. Otros enlaces a la izquierda que se utilizan para regresar, a excepción de una pila que usa un sistema LIFO (Last In First Out). Como lo indican sus siglas en inglés es el último en entrar, es el primero en salir, entonces se podría decir que el último elemento en entrar desplaza a todos los demás posicionándose en la cabeza y así sucesivamente. Cola es un sistema FIFO (First In First Out) donde el primero en entrar es el último en salir, permitiendo un funcionamiento conforme al orden de llegada.

- Permitiendo una estructura sólida para la accesibilidad, monitorización, escalabilidad, manejo de datos de forma adecuada y una seguridad de la información completa. Ya que el sistema permite un acceso a sus funcionamientos de una forma práctica para adaptarse a cualquier situación con una monitorización eficiente para hacer el proceso de seguimiento del rendimiento para identificar y solucionar problemas yendo de la mano con la escalabilidad para adaptarse y evolucionar como proyecto para un mayor número de usuarios y a una mayor demanda de servicios con un manejo de datos eficiente y práctico permitiendo los procesos que se utilizan para el manejo de datos de forma eficiente haciendo la debida protección de los datos sensibles de las personas, protegiéndolos contra errores, daños o accesos no autorizados.
- Y todo esto, teniendo siempre presente el Diseño centrado en el usuario (UCD) para garantizar así la completa comprensión y funcionalidad del programa, donde se basa en comprender las necesidades, expectativas y comportamientos de los usuarios para crear productos y servicios que se ajusten a ellos.

8. Marco Legal

El marco legal que rige el desarrollo y la implementación del sistema para la atención médica desempeña un papel crucial en la garantía de la calidad, la seguridad y la protección de los derechos de los pacientes y los profesionales de la salud. En un entorno delicado y sensible como el sector de la salud, el cumplimiento de las leyes y regulaciones es esencial para salvaguardar la integridad y la confianza en los servicios prestados.

El marco legal presente en este documento abarca una amplia gama de áreas, desde la protección de datos personales y la confidencialidad médica hasta las normativas específicas para el desarrollo del software en el ámbito de la salud. Cada una de estas áreas tiene como objetivo asegurar que el sistema cumpla con los estándares éticos y legales necesarios para el correcto funcionamiento.

Al comprender y adherirse a nuestro marco legal aplicable, los desarrolladores y proveedores de servicios de salud pueden mitigar riesgos legales, evitar sanciones y, lo que es más importante, proteger la privacidad y derechos de los pacientes. En esta introducción al marco legal, se explorarán las principales leyes y regulaciones que deben tenerse en cuenta en el desarrollo e implementación de un sistema relacionado con la salud, así como las implicaciones prácticas de cumplir con estas normativas en la práctica clínica y administrativa.

1. Confidencialidad médica

La confidencialidad médica es importante para los pacientes y sus profesionales de la salud. Con solo unas pocas excepciones, todo lo que hable con su médico debe, por ley, mantenerse en privado entre ambas partes como usuario y médico y la organización para la que trabajan o donde se emplea el servicio. Esto también se conoce como confidencialidad médico-paciente.

2. Privacidad en la atención sanitaria

Bajo las leyes de privacidad en la atención sanitaria, los pacientes tienen el derecho fundamental a controlar quién puede acceder a su información médica, cómo se utiliza y divulga, y con qué fines. Esta protección se extiende a todos los aspectos relacionados con la atención médica, incluyendo el almacenamiento de registros médicos, la comunicación con proveedores de servicios de salud, y la participación en investigaciones médicas.

3. Cumplimiento Normativo

Hay que asegurar que el sistema cumpla con todas las regulaciones y normativas aplicables en el ámbito de la salud, incluyendo leyes locales, regionales y nacionales, así como estándares internacionales si son relevantes. Como lo dice La Ley Estatutaria De Salud (Ley 1751 de 2015). Esta ley establece los principios y los derechos relacionados con la salud en Colombia, incluyendo, la participación ciudadana en el sistema de salud, entre otros aspectos fundamentales. Esto implica asegurar que el sistema cumpla con los estándares de calidad y seguridad requeridos, protegiendo los derechos de los pacientes y promoviendo una atención médica equitativa y accesible para todos los ciudadanos colombianos.

4. Ley General de Salud (Ley 1438 de 2011)

La Ley 1438 de 2011 introduce importantes cambios en el sistema de salud colombiano, con el objetivo de mejorar la calidad, la eficiencia y la equidad en la prestación de servicios de salud. Entre sus disposiciones, la ley establece la creación del Sistema General de Seguridad Social en Salud (SGSSS), que tiene como objetivo garantizar el acceso universal a los servicios de salud y proteger el derecho a la salud de todos los ciudadanos colombianos.

Además, la Ley 1438 de 2011 aborda temas como la regulación de la prestación de servicios de salud, la gestión de los recursos en el sistema de salud, la promoción de estilos de vida saludables, la participación ciudadana en la toma de decisiones en salud, entre otros aspectos relevantes para el funcionamiento del sistema de salud en Colombia.

5. Ley de protección de datos personales (Ley 1581 de 2012)

Esta ley establece los principios y disposiciones para la protección de los datos personales en Colombia, incluyendo la recolección, almacenamiento, uso, circulación y supresión de dichos datos, lo cual es especialmente relevante en el contexto de la gestión de información de pacientes en el Software.

6. Resolución 1995 de 1999 del Ministerio de Salud y protección Social

Esta resolución establece los requisitos técnicos y operativos que deben cumplir los establecimientos de salud en Colombia, incluyendo aspectos relacionados con la gestión de la información de pacientes y la interoperabilidad de sistemas de información en salud.

7. Normas internacionales de información de Salud (CIE-10, CIE-11)

Estas normas internacionales establecen un marco para la codificación de enfermedades, procedimientos y otros conceptos relacionados con la salud, lo cual puede ser relevante para la interoperabilidad y la estandarización de datos en el software.

8. Protección de datos personales

Garantizar el cumplimiento de las leyes de protección de datos personales, como el reglamento General de protección de datos (GDPR) en la Unión Europea o leyes similares en otras jurisdicciones, para proteger la información personal de los pacientes.

9. Consentimiento informado

Establecer procedimientos para obtener el consentimiento informado de los pacientes para el uso y la divulgación de su información médica, de acuerdo con las leyes y regulaciones aplicables.

10. Seguridad de la información

Implementar medidas de seguridad robustas para proteger la información de salud de los pacientes contra accesos no autorizados, divulgación indebida, alteración o destrucción, de acuerdo con las prácticas y estándares de seguridad de la industria.

11. Responsabilidad legal

Definir claramente las responsabilidades legales de todas las partes involucradas en el desarrollo, implementación y uso del sistema, incluyendo el proveedor del sistema, los profesionales de la salud, y los usuarios finales.

12. Registro y auditoría

Establecer procedimientos para el registro y la auditoría de todas las actividades relacionadas con el sistema, incluyendo el acceso a la información de salud de los pacientes, para garantizar la transparencia y la rendición de cuentas.

13. Notificación de brechas de seguridad

Establecer protocolos para la notificación oportuna de cualquier brecha de seguridad que pueda afectar la privacidad o seguridad de la información de salud de los pacientes, de acuerdo con las leyes y regulaciones aplicables.

14. Ley de 1551 de 2015

Una de las leyes fundamentales que respaldan estos principios es la Ley Estatutaria de Salud en Colombia, Esta ley establece los principios y derechos relacionados con la salud en el país, incluyendo la garantía de acceso a servicios de salud de calidad, la protección de los derechos de los pacientes, la participación ciudadana en el sistema de salud, entre otros aspectos relevantes.

Además, el Código de Ética Médica Colombiano, establecido por la Asociación Médica Colombiana, también aborda principios éticos y estándares de práctica clínica que deben seguir los profesionales de la salud en el país. Este código incluye disposiciones sobre el consentimiento informado, la confidencialidad médica, la relación médico-paciente, entre otros aspectos importantes para garantizar una atención médica ética y segura.

15. Ley 39 de 1990

La ley colombiana que establece que no pueden existir dos personas con el mismo documento de identidad es la Ley 39 de 1990, específicamente en su artículo 11. Esta ley regula el Registro Civil y establece los principios y procedimientos para la identificación de las personas en Colombia. El artículo 11 de esta ley establece que "en ningún caso se asignará a una persona un número de identificación que haya sido asignado previamente a otra". Esto significa que el número de identificación, como el documento de identidad (cédula de ciudadanía o tarjeta de identidad), debe ser único para cada individuo en Colombia.

16. Ley 1480 de 2011

Conocida como el Estatuto del Consumidor, que establece los derechos y deberes de los consumidores en Colombia y regula las relaciones de consumo, incluyendo las transacciones comerciales relacionadas con servicios de salud. Aunque esta ley no aborda específicamente el tema del retorno del dinero en el contexto de servicios médicos, sí establece principios generales de protección al consumidor. Como se enfatiza a continuación, la devolución de dinero por servicios médicos en Colombia está sujeta a disposiciones generales de protección al consumidor, como las establecidas en la Ley 1480 de 2011, conocida como el Estatuto al Consumidor. Esta ley establece los derechos y deberes de los consumidores Colombianos y regula las relaciones de consumo, incluyendo transacciones comerciales relacionadas con servicios de salud. Incluyendo el reembolso del dinero.

17. Consentimiento informado para exámenes Médicos

El consentimiento informado es esencial en la relación médico-paciente y la relación de exámenes médicos. Antes de cualquier procedimiento, es necesario que el usuario comprenda los riesgos, beneficios y alternativas. Según la Ley estatutaria de Salud (Ley 1751 de 2015) y la Ley General de Salud (Ley 1438 de 2011), los pacientes tienen derecho a decidir sobre su salud. Los profesionales de la salud deben obtener el consentimiento del paciente de forma voluntaria y libre. Nos comprometemos a seguir estas regulaciones para proteger los derechos de nuestros usuarios y garantizar una atención médica ética y segura.

9. Diseños

Los diseños en este proyecto desempeñan un papel crucial en el éxito del proyecto Aurora, ya que proporciona la estructura y el plan detallado para la implementación del sistema de gestión de citas médicas. Estas etapas de diseño son fundamentales para traducir los objetivos del proyecto en soluciones tangibles y funcionales. A través de los diseños, se establecerán los fundamentos para una implementación eficaz y eficiente del sistema, garantizando que cumpla con las necesidades y expectativas de los usuarios finales.

- Aspectos clave de los diseños en el proyecto

Las etapas específicas de diseño, aborda diversas etapas de diseño, incluyendo el diseño de la arquitectura del sistema, el diseño de la interfaz de usuario y el diseño de la persistencia de la información y como se transmitirá la estructura del programa con los miembros del equipo haciendo el uso de diagramas de clase, entre otros. Cada etapa será cuidadosamente planificada y ejecutada para garantizar la coherencia y funcionalidad del sistema en su conjunto.

Donde el enfoque en la usabilidad y experiencia del usuario se pondrá un énfasis especial en la usabilidad y la experiencia del usuario durante el diseño del sistema. Se implementarán principios de diseño centrado en el usuario para crear una interfaz intuitiva y fácil de usar que mejore la experiencia del paciente en la gestión de citas médicas.

Por otro lado, el enfoque en la escalabilidad y flexibilidad donde se diseñarán un sistema que sea escalable y flexible para adaptarse a futuras necesidades y cambios en el entorno de atención médica. Se considerarán cuidadosamente aspectos de escalabilidad y flexibilidad para asegurar la viabilidad a largo plazo del sistema.

9.1 Modelo de datos

El modelo de datos que se presenta a continuación es el resultado del análisis que se hizo a las necesidades y desafíos identificados en el proceso de implementación de un sistema de gestión de pacientes. Este modelo se ha desarrollado con el objetivo de proporcionar una estructura sólida y eficiente para la gestión de citas médicas y exámenes y los demás requisitos solicitados por el cliente, con el fin de mejorar la calidad y la accesibilidad de la atención médica para los residentes de Bucaramanga.

Para satisfacer de forma efectiva en el entorno de salud que es dinámico y exigente, la gestión eficiente, pago de citas, cancelación, registro de autorizaciones y exámenes es fundamental para garantizar la satisfacción de los usuarios y la eficacia del sistema de salud en su conjunto.

Como objetivo, este modelo de datos presentado tiene como objetivo principal proporcionar una estructura coherente y bien definida para la gestión de datos relacionados con citas médicas y exámenes en el centro de salud. Permitiendo así una eficiente gestión, facilitando la programación, la coordinación y el seguimiento de las actividades relacionadas. Además, el modelo se diseñará teniendo en cuenta la escalabilidad y flexibilidad, para adaptarse a las necesidades cambiantes del sistema, garantizando así su viabilidad a largo plazo.

Entidades Principales

1. **Paciente/Usuario:** Representa a los individuos que solicitan citas médicas y exámenes.
 - **Atributos:** Nombre, apellido, identificación, edad, teléfono, contraseña.
2. **Cita Médica:** Representa una cita programada entre un paciente y un médico.
 - **Atributos:** Nombre, apellido, identificación, identificador, ticket, costo, doctor, estado, especialidad, asistencia, motivo.
3. **Autorización:** Representa la autorización para los exámenes solicitados para un paciente.
 - **Atributos:** Nombre, apellido, identificación, identificadorde la cita, ticketde la cita, precio, tipoExamen, identificadorde autorizacion, descripción, Estado, ticketDeAutorizacion.

4. **Examen Médico:** Representa un examen médico solicitado por un paciente para ser atendido en la categoría de exámenes. Para esta solicitud necesita previamente una cita médica de tipo control para gestionar que su estado de salud no se verá afectado después de los exámenes y una autorización exitosa para ser atendido con éxito.
 - **Atributos:** nombreUsuario, apellidoUsuario, identificacionDeExamen, ticketDeExamen, costo, tipoExamen, descripción, estadoDePago, asistencia

Relaciones

1. **Relación Paciente - Cita Médica:** Un paciente puede tener múltiples citas médicas independientemente de su especialidad, motivo u doctor. Y cada cita médica está asociada a un único paciente.
2. **Relación Pago de Cita – Cita Médica:** Usted para ser atendido necesita previamente pagar su cita médica a no ser que se trate de un control que por defecto viene pago por ser gratuito.
3. **Relación Cita Médica - autorización:** Un paciente puede tener múltiples autorizaciones médicos, y cada autorización está asociada a un único paciente con su debida cita de control para apoyarnos en el marco legal evitando posibles situaciones donde se afecte la vida del usuario por practicar exámenes donde no estaba completamente bien su estado para ser tratado con base a un examen.
4. **Autorización - Examen Médico:** Un paciente puede tener múltiples exámenes médicos, y cada examen médico está asociado a un único paciente. Para ser atendido con un examen usted necesita una autorización exitosa para poder ser atendido con un examen.

- Consideraciones Adicionales

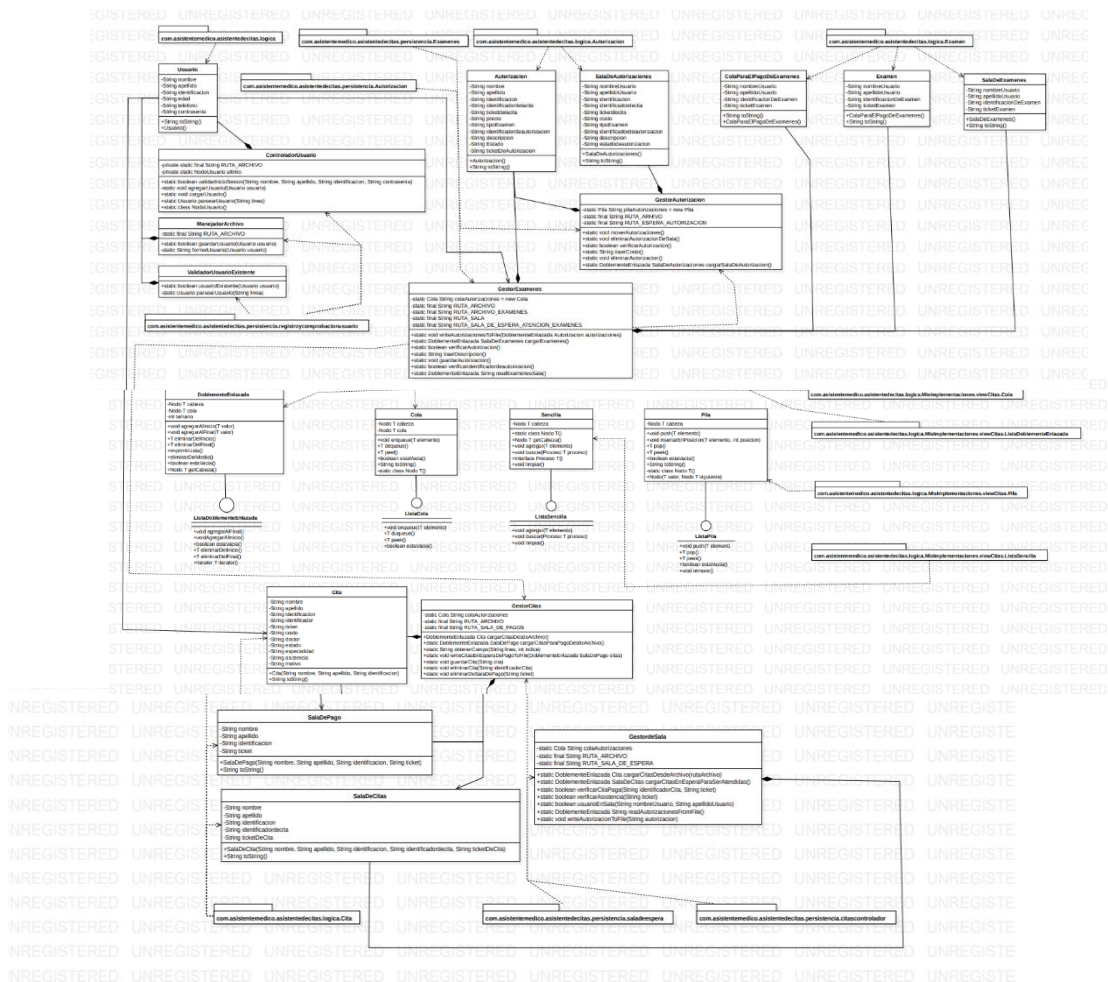
- Se podría agregar una tabla para gestionar los médicos, incluyendo su nombre, especialidad.
- Se puede incluir tablas adicionales para la visualización de las citas , autorizaciones o exámenes relacionados con el usuario.
- Se pueden establecer restricciones y reglas de negocio para garantizar la integridad y la consistencia de los datos en el modelo, como términos y condiciones.

9.2 Diagramas

Los diagramas en este proyecto desempeñan un papel fundamental en el desarrollo y la implementación del proyecto Aurora, proporcionando herramientas visuales que permiten una comprensión clara y estructurada de los diferentes aspectos del sistema de gestión de citas médicas. En particular, los diagramas de clase, caso de uso y de secuencia son elementos esenciales que ayudan a definir la arquitectura, la funcionalidad y la interacción del sistema de manera precisa y detallada.

9.2.1 Diagrama de clase

El diagrama de clase presente permite representar nuestra estructura estática del sistema, mostrando las clases, atributos, métodos y relaciones entre ellas. En el contexto del proyecto Aurora, el diagrama de clase es crucial para definir la organización y la interacción de los diferentes componentes del sistema, incluyendo entidades principales como usuario, citas médicas, autorizaciones y exámenes médicos. Además, los diagramas de clase proporcionan una base sólida para el diseño de la persistencia de datos y la implementación del modelo de datos.

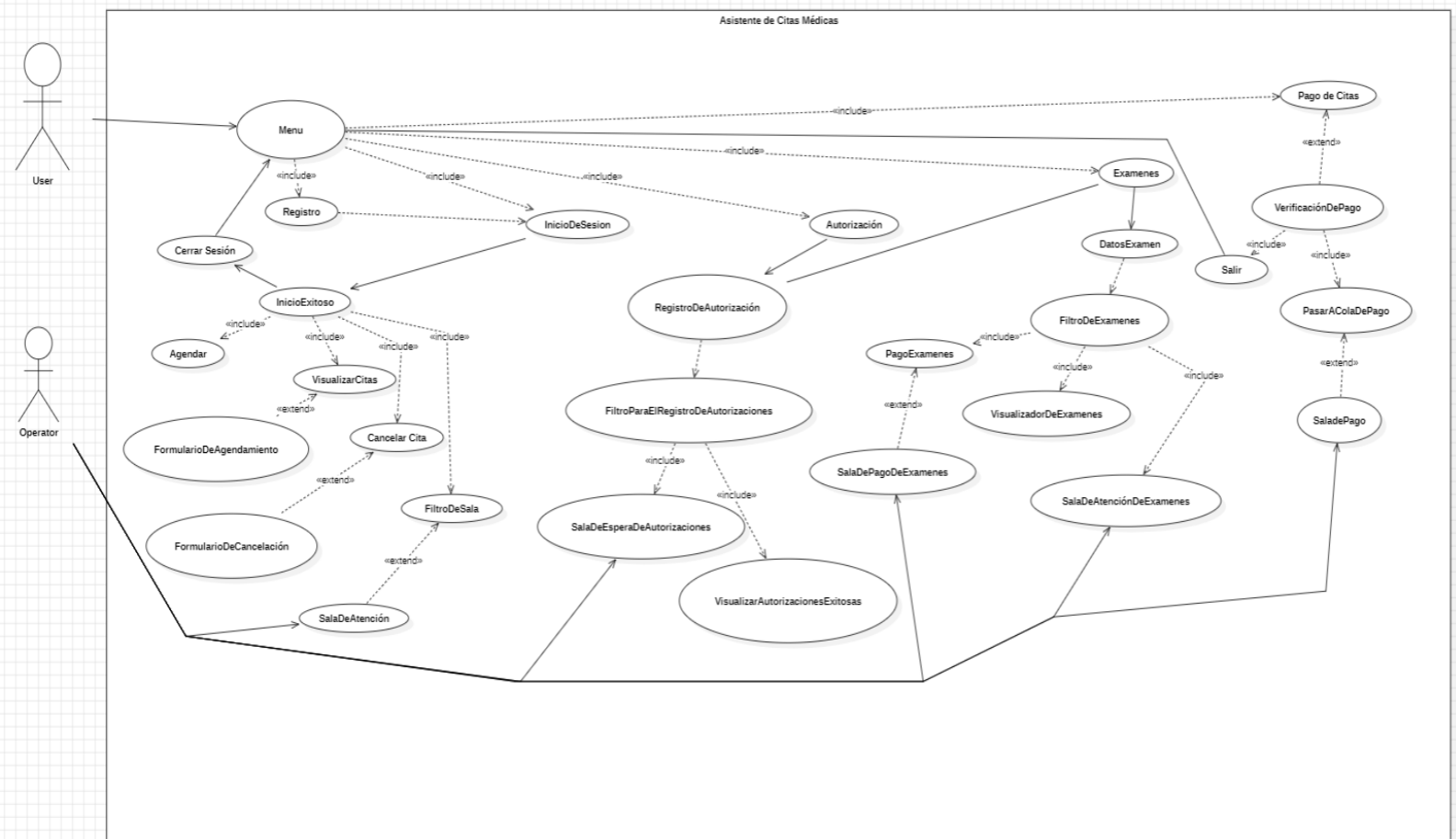


9.2.2 Diagrama de casos De Uso

El presente diagrama de casos de uso muestra una serie de casos de uso que describen las funcionalidades y los procesos clave del sistema. Los actores externos que interactúan con el sistema se representan en el borde del diagrama con los nombres de User y Operator. Mientras que los casos de uso se muestran en el interior de este.

Cada caso de uso describe una interacción específica entre un actor y el sistema, identificando las acciones que se realizan y los resultados esperados. Los casos de uso se organizan de manera lógica y se agrupan en función de las funcionalidades relacionadas, lo que permite una comprensión clara y estructurada de las capacidades del sistema.

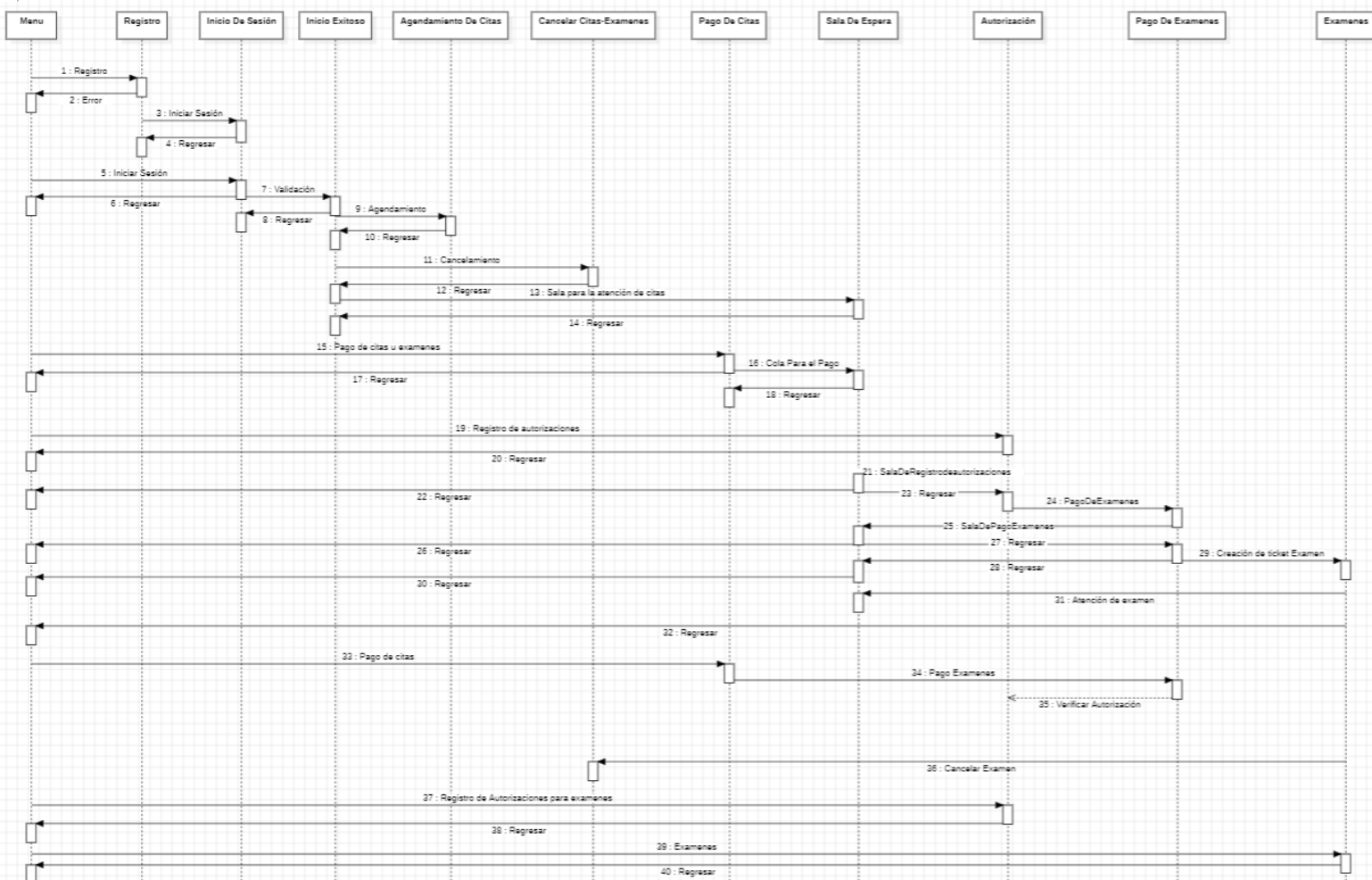
El objetivo del presente diagrama de casos de uso es proporcionar una descripción completa y detallada de las funcionalidades del sistema desde la perspectiva del usuario y el operador. Esto incluye identificar las diversas tareas que los usuarios puedan realizar y las tareas que el operador puede intervenir, los requisitos funcionales del sistema y las interacciones entre los usuarios y el sistema. Dando así un funcionamiento como herramienta de comunicación efectiva entre los diferentes interesados en el proyecto, permitiendo una alineación clara de las expectativas y requisitos del sistema.



9.2.3 Diagrama de secuencia

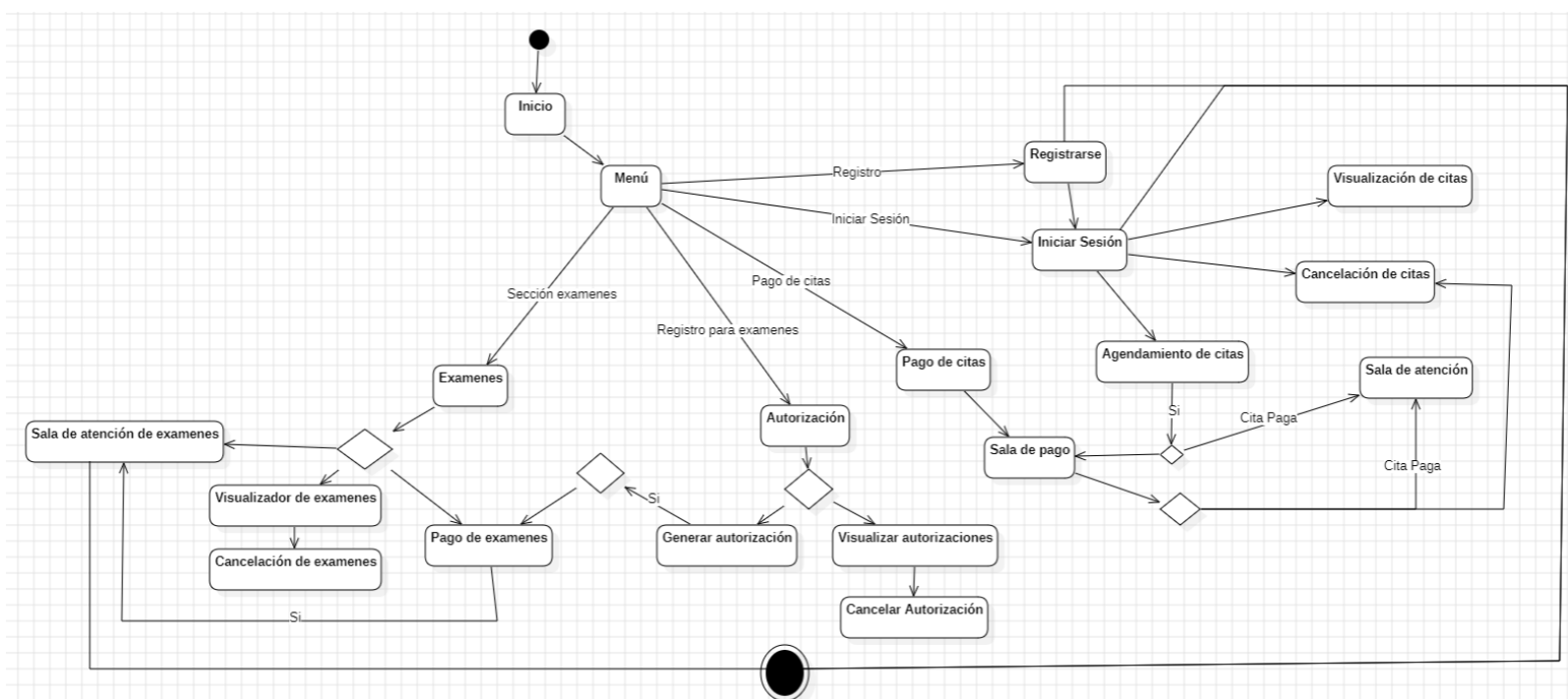
El diagrama de secuencia de esta sección muestra una serie de eventos y las interacciones que ocurren entre los objetos del sistema en respuesta a los eventos. Los objetos se representan en el diagrama, junto con las líneas de mensaje que indican la comunicación entre ellos. Estas líneas de mensaje muestran el orden y la dirección en la se envían los mensajes entre los objetos, así como cualquier dato adicional que se transmita.

Cada interacción se representa secuencialmente en el diagrama, lo que permite visualizar el flujo de control y los intercambios de información entre los objetos del sistema. El objetivo principal y específico de este diagrama de secuencia es proporcionar la representación detallada y práctica y clara de la dinámica del sistema, mostrando cómo los diferentes objetos colaboran entre sí para lograr ciertos objetivos. Esto incluye identificar los procesos y las interacciones clave del sistema, así como comprender la lógica y el flujo de control detrás de estas interacciones. El presente diagrama cumple la funcionalidad para la identificación de posibles problemas y errores en el diseño del sistema, así como para comunicar de manera efectiva requisitos y especificaciones.



9.2.4 Diagrama de actividades

El diagrama de actividades presente visualiza el flujo de trabajo dentro del sistema, mostrando las actividades que tiene disponible el usuario y las relaciones entre ellas con flechas. Cada actividad representa una acción en el proceso como “Agendamiento de cita” o “Cancelar Examen”. Este diagrama ayuda a comprender cómo las actividades se conectan entre sí cómo fluye el proceso en el sistema, facilitando la identificación de mejoras y de forma práctica.



9.3 Actividades

ID	ACTIVIDAD	DESCRIPCIÓN	PRIORIDAD
1	Análisis	Se realiza la respectiva investigación sobre el sistema que se va a desarrollar, sus funcionalidades, requerimientos, para una mejor organización en el momento de realizar el proyecto	ALTA
2	Primera edición del documento	Se comenzó a editar el documento las primeras características como introducción, situación problema, pregunta problema, justificación y objetivos.	MEDIA
3	Metodología y documentación	Se conversa acerca de la metodología a emplear para avanzar en el proyecto, la cual fue la metodología espiral, una vez decidida se comienza a aplicar en la realización del proyecto y se agrega a la documentación,	ALTA
4	Segunda edición del documento	Se agregan los marcos, requerimientos, cronograma y bibliografía, de esta forma se finaliza los aspectos necesarios para la primera entrega del documento	ALTA
5	Primer avance del sistema de citas médicas	El primer avance consistió en la creación de las interfaces para cada una de las funcionalidades a implementar.	MEDIA
6	Segundo avance del sistema de citas médicas	Se implementan funcionalidades principales y adicionales, como sala de espera, autorización de exámenes, pago de citas, registro de autorizaciones, atención de exámenes, cancelación de citas y exámenes.	ALTA
7	Prueba de las funcionalidades implementadas	Se realizan pruebas para comprobar la funcionalidad del código y mitigar el riesgo de continuar con la creación del sistema, para esto se hace la prueba de cada funcionalidad por separado.	MUY ALTA
8	Tercer avance del sistema de citas médicas	Se empezaron a implementar otros requerimientos documentados como la interfaz para el inicio de sesión, configuración y selección del tipo de cita, confirmación de la cita, modificación de la cita designada. Buscando una sincronización final entre funcionalidad y algo práctico.	ALTA

9.4 Interfaces

Las interfaces juegan un papel fundamental tanto visual como en estructura organizada de código en el proyecto Aurora al proporcionar el punto de interacción entre los usuarios y el sistema de gestión de citas médicas. Estas interfaces son la cara visible del sistema, permitiendo a los usuarios acceder y utilizar las diferentes funcionalidades de manera intuitiva y eficiente. En el contexto de este proyecto, las interfaces son elementos clave que facilitan la comunicación y la interacción entre los usuarios y el sistema, asegurando una experiencia de usuario óptima y satisfactoria.

La importancia de una correcta aplicación de interfaces y patrones de diseño son una parte integral de la implementación del sistema de gestión de citas médicas en el proyecto Aurora. Su importancia radica en varios aspectos clave como son:

- La facilidad de la usabilidad, donde esto se puede obtener por medio de interfaces bien diseñadas que permiten a los usuarios interactuar de manera fácil y eficiente con el sistema, facilitando la navegación y el acceso a las funcionalidades necesarias para gestionar citas médicas y exámenes.
- Optimizan la productividad, ya que una interfaz eficiente y bien diseñada ayudan a agilizar los procesos de gestión, lo que resulta en una mayor productividad y eficiencia tanto para los usuarios como para el personal médico.
- Garantizan la accesibilidad, ya que interfaces accesibles y adaptativas aseguran que el sistema pueda ser utilizado por una amplia gama de usuarios, incluyendo aquellos con diferentes niveles de habilidad y capacidades.
- Y uno de los factores más importantes es el mejoramiento de la experiencia del usuario, donde si es una interfaz intuitiva y atractiva contribuye a una experiencia positiva, aumentando la satisfacción y la aceptación del sistema por parte de los usuarios finales, como pacientes y personal médico.

9.4.1 Vistas de Diseño

Las vistas de diseño desempeñan uno de los papeles más importantes en el proyecto ya que son esenciales ya que actúan como el punto de interacción principal entre los usuarios y el sistema de gestión de citas médicas. Estas interfaces representan la interfaz visual a través de la cual los usuarios pueden acceder y utilizar las diversas funcionalidades proporcionadas por el sistema. Son la ventana a través de la cual los usuarios pueden interactuar de manera efectiva y eficiente con las capacidades que ofrece el sistema.

En el contexto aurora, las interfaces son más que simples representaciones visuales; también están diseñadas para proporcionar diferentes puntos de vista y una estructura organizada en el código subyacente. Esto garantiza una coherencia tanto en la apariencia visual como en la funcionalidad del sistema, lo que contribuye a una experiencia de usuario cohesiva y satisfactoria.

Estas interfaces están cuidadosamente diseñadas para ser intuitivas y fáciles de usar, lo que permite a los usuarios navegar sin problema a través de las diferentes características del sistema. Se han considerado principios de diseño de experiencia de usuario (UX) para garantizar que la interacción con el sistema sea lo más fluida y natural posible.

1. Ventana Principal/LOBBY

Frame de ventana Principal, este Frame es la pieza central de la interfaz y por donde la entrada del programa al ejecutarse es el primero en iniciar. Es la pieza central de la interfaz de usuario, que proporciona acceso a las principales funciones y características del sistema. Su diseño minimalista y elegante incorpora una paleta de colores que incluye el tono de gris, blanco, azul marino y verde suave. También un clip de bienvenidos para hacerlo de una forma dinámica y una imagen propia de logo y otra imagen que llena un vacío, creando una atmósfera serena y moderna. Además de ser estéticamente atractivo, este Frame está diseñado



2. Ventana Registro/Inicio De Sesión

En sección se proporciona a los usuarios la capacidad de crear una cuenta en el sistema, permitiéndoles acceder a todas las funciones y características disponibles. Con una paleta igual a la del Principal ya que esa paleta se manejó en todo el sistema. En este proceso de registro se solicita parámetros al usuario como se notan en la imagen y un aceptó de términos y condiciones para ir de la mano con el marco legal y su correcta aplicación con base a la ley del trato de datos sensibles del usuario. Aparte en la sección de inicio de sesión siguiendo la misma estética solicitando 4 parámetros para acceder a funciones que se especificarán a continuación.

Asistente de Citas Médicas

Exámenes Pago de Citas Autorización Iniciar Sesión Lobby

Registro

Nombre(S):

Apellidos:

N° Identificación:

Edad:

Número de Teléfono:

Contraseña:

Repetir Contraseña:

☐ Mostrar Contraseña

Términos y Condiciones

Por favor lea a continuación los terminos y condiciones con base al tratamiento de datos lea atentamente a continuación:

Aceptación de los Términos: Al acceder o utilizar la Aplicación de cualquier manera, usted acepta estar legalmente obligado por estos Términos y nuestra Política de Privacidad. Si no está de acuerdo con estos Términos, por favor no utilice la Aplicación.

Uso Aceptable: Usted se compromete a utilizar la Aplicación únicamente para fines legales y de acuerdo con estos Términos. No debe utilizar la Aplicación de ninguna manera que pueda dañar, deshabilitar, sobrecargar o comprometer nuestra infraestructura de red o seguridad.

☐ Aceptar Términos

Registrar Limpiar

Derechos de autor © 2024 Asistente de Citas Médicas. Todos los derechos reservados. Prohibida la reproducción total o parcial de este sitio web, incluidos textos, imágenes y diseño, sin la autorización previa por escrito de Asistente de Citas Médicas. Email: asistentedecitasmedicas@gmail.com

ASISTENTE DE CITAS MÉDICAS

Exámenes Pago de Citas Autorización Lobby Registrarse

Iniciar Sesión

Nombre(S):

Apellidos:

N° Identificación:

Contraseña:

☐ Mostrar Contraseña

Ingresar Limpiar

Términos y Condiciones

Por favor lea a continuación los terminos y condiciones con base al tratamiento de datos lea atentamente a continuación:

Aceptación de los Términos: Al acceder o utilizar la Aplicación de cualquier manera, usted acepta estar legalmente obligado por estos Términos y nuestra Política de Privacidad. Si no está de acuerdo con estos Términos, por favor no utilice la Aplicación.

Uso Aceptable: Usted se compromete a utilizar la Aplicación únicamente para fines legales y de acuerdo con estos Términos. No debe utilizar la Aplicación de ninguna manera que pueda dañar, deshabilitar, sobrecargar o comprometer nuestra infraestructura de red o seguridad.

Derechos de autor © 2024 Asistente de Citas Médicas. Todos los derechos reservados. Prohibida la reproducción total o parcial de este sitio web, incluidos textos, imágenes y diseño, sin la autorización previa por escrito de Asistente de Citas Médicas. Email: asistentedecitasmedicas@gmail.com

3. Entrada Home de inicioExitoso

Esta página de inicio da la bienvenida a los usuarios de forma exitosa después de iniciar sesión en el sistema. Al acceder con éxito, los usuarios son dirigidos a la página principal, donde se les presentan una serie de nuevas opciones y funcionalidades para explorar.

Las opciones disponibles después de iniciar Sesión exitosamente incluyen:

1. Agendamiento de citas
2. Cancelación de citas
3. Visualizar citas a nombre del usuario (Historia Clínica)
4. Sala de espera

Como diseño adicional se aplicó que fuera un espacio personalizado como se puede apreciar teniendo una persistencia de quien es el usuario que está haciendo las acciones en este caso Samuel Arango Diaz.

The screenshot displays the 'Asistente de Citas Médicas' web application. The header features a medical icon, the title 'Asistente de Citas Médicas', and navigation buttons: 'Exámenes', 'Pago de Citas', 'Autorización', 'Lobby', and 'Registrarse'. The main content area is divided into two sections. The left section, titled 'Especialidad Requerida', lists medical specialties with radio buttons: 'Medicina General', 'Pediatría', 'Ginecología', 'Cardiología', and 'Dermatología'. Below this is the 'Motivo de Cita' section with radio buttons for 'Valoración', 'Asesoramiento', 'Control', and 'Exámen'. At the bottom of the left section are 'Agendar' and 'Salir' buttons. The right section, titled 'Espacio de', shows the user's name 'Samuel Arango Díaz' and three buttons: 'Sala de espera', 'Cancelar Cita', and 'Visualizar Citas'. A footer note states: 'Recuerde por favor que otra especialidad aparte de medicina general tendra un aumento de 50000 pesos incluyendo IVA y que los controles son gratuitos'. The bottom of the page contains copyright information and the email 'Email: asistentedecitasmedicas@gmail.com'.

4. Agendamiento de citas

El agendamiento de citas como se aprecia proporciona a los usuarios la capacidad de programar nuevas citas de manera rápida y eficiente. Esta es una parte que se encuentra anterior a la confirmación de la cita, pasando por diferentes filtros inicialmente donde el usuario debe seleccionar la especialidad y el motivo luego el doctor generando así el ticket con base a lo seleccionado el identificador de la cita y el costo respectivo. Inicialmente para todas las citas excepto para los controles el botón de pagar estará inicialmente desactivado ya que obligatoriamente se debe pasar a pago de citas para hacer el respectivo pago excepto para los controles donde este flag está activado de pago. A continuación, se visualizará también el agendamiento exitoso.

Asistente de Citas Médicas

Exámenes Pago de Citas Autorización Lobby Registrarse

Especialidad Requerida

- ☐ Medicina General
- ☐ Pediatría
- ☐ Ginecología
- ☐ Cardiología
- ☐ Dermatología

Motivo de Cita

- ☐ Valoración
- ☐ Asesoramiento
- ☐ Control

Volver Salir

Doctores Disponibles

☐ Doctor Arango ☐ Doctor Juan Carlos ☐ Doctor Jelson

Identificador de Cita 10909

Ticket GIA001

Costo 150000

IVA Incluido

AGENDAR PAGAR

Espacio de Samuel Arango Diaz

Sala de espera

Cancelar Cita

Visualizar Citas

Derechos de autor © 2024 AsistenteDeCitasCA. Todos los derechos reservados. Prohibida la reproducción total o parcial de este sitio web, incluidos textos, imágenes y diseño, sin la autorización previa por escrito de AsistenteDeCitasMedicasCA. Email: asistentedecitasmedicas@gmail.com

Asistente de Citas Médicas

Exámenes Pago de Citas Autorización Lobby Registrarse

Especialidad Requerida

- ☐ Medicina General
- ☐ Pediatría
- ☐ Ginecología
- ☐ Cardiología
- ☐ Dermatología

Motivo de Cita

- ☐ Valoración
- ☐ Asesoramiento
- ☐ Control

Volver Salir

Doctores Disponibles

☐ Doctor Arango ☐ Doctor Juan Carlos ☐ Doctor Jelson

Identificador de Cita 10909

Ticket GIA001

Costo 150000

IVA Incluido

PAGO

AGENDAMIENTO EXITOSO DE CITA

Espacio de Samuel Arango Diaz

Sala de espera

Cancelar Cita

Visualizar Citas

Derechos de autor © 2024 AsistenteDeCitasCA. Todos los derechos reservados. Prohibida la reproducción total o parcial de este sitio web, incluidos textos, imágenes y diseño, sin la autorización previa por escrito de AsistenteDeCitasMedicasCA. Email: asistentedecitasmedicas@gmail.com

5. Visualización de citas

Esta página muestra una lista completa de todas las citas programadas a nombre del usuario actual. Proporciona una visión general de las citas próximas y pasadas, permitiendo al usuario acceder fácilmente a la información detallada de cada Cita.

Características clave:

1. Lista de citas, muestra todas las citas programadas en formato sencillo de entender con detalles exactos.
2. Filtros de búsqueda ofrece solamente las citas con base al usuario que solicitó visualizar.
3. Detalles y continúan las opciones de gestión.

The screenshot shows a web application titled "Asistente de Citas Médicas". At the top, there is a navigation bar with a medical icon and several tabs: "Exámenes", "Pago de Citas", "Autorización", "Lobby", and "Registrarse". The main content area features a table with columns for patient information, appointment details, and payment status. The table contains two rows of data for a patient named Samuel Arango Diaz. To the right of the table, there is a sidebar with a user profile section for "Samuel Arango Diaz" and buttons for "Sala de espera" and "Cancelar Cita". At the bottom of the main area, there are buttons for "Salir", "Volver", and "Visualizar". A small text note below the "Visualizar" button says "Da un click al boton visualizar para cargar tus citas". The footer contains copyright information and an email address.

Nombre	Apellido(s)	Ident.	Iden.Cta	Ticket	Costo	Doctor	Pago?	Especialidad	Asistida?	Motivo
Samuel	Arango Diaz	1116774764	9650	PV001	150000	Doctor Jelson	Paga	Pediatría	Asistida	Valoración
Samuel	Arango Diaz	1116774764	10909	GIA001	150000	Doctor Jelson	No Paga	Ginecología	No asistida	Asesoramiento

Salir Volver Visualizar

Da un click al boton visualizar para cargar tus citas

Espacio de Samuel Arango Diaz Sala de espera Cancelar Cita

Derechos de autor © 2014 AsistenteCitasCA. Todos los derechos reservados. Prohibida la reproducción total o parcial de este sitio web, incluidos textos, imágenes y diseño, sin la autorización previa por escrito de AsistenteCitasMedicasCA. Email: asistentecitasmedicas@gmail.com

6. Cancelación de citas

Aquí se centra en proporcionar al usuario la capacidad de cancelar citas médicas de manera eficiente y sin complicaciones. Aquí hay puntos a resaltar como que se le piden 4 datos respecto a la cita a cancelar que son el identificador, especialidad, motivo y ticket el proceso de cancelación simplificado asegura de que el proceso de cancelación sea lo más sencillo posible para el usuario. Implicando una interfaz intuitiva fácil de navegar que permite al usuario cancelar una cita con unos pocos clics o toques. Cabe resaltar que usted no puede cancelar citas a las que donde ya ha asistido ya que no tendría lógica eliminar ese historial. Pero si pudiera cancelar citas pagas, no pagas, no asistidas. El manejo de excepciones del ticket ya que por ejemplo uno de los requerimientos era tener en cuenta cuantas citas ya se han confirmado para crear el ticket. Pues continuaría su flujo normal del último ticket.

The screenshot displays the 'Asistente de Citas Médicas' web application interface. At the top, there is a navigation bar with links: 'Exámenes', 'Pago de Citas', 'Autorización', 'Lobby', and 'Registrarse'. The main content area is divided into several sections:

- Especialidad Requerida:** A list of medical specialties with radio buttons. 'Ginecología' is selected.
- Motivo de Cita:** A list of reasons for the appointment with radio buttons. 'Valoración' is selected.
- Identificador de Cita:** A text input field containing '456815'.
- Ticket:** A text input field containing 'GC004'.
- Confirmación:** A 'CONFIRMAR CANCELACION' button.
- Recuerde:** A note stating 'Recuerde que al cancelar la cita se le regresara el 70% del dinero'.
- Acciones:** 'Volver' and 'Salir' buttons.
- Usuario:** 'Espacio de Samuel Arango Diaz'.
- Salida:** 'Sala de espera' and 'Visualizar Citas' buttons.

At the bottom, there is a footer with copyright information and an email address: 'Email: asistentecitasmedicas@gmail.com'.

7. Sala de espera para atención de citas

La sala de espera es un componente crucial en el proceso de atención de citas médicas, ya que proporciona un espacio virtual donde los pacientes pueden esperar su turno para ser atendidos. La interfaz es intuitiva para hacer fácil su uso y accesible para los pacientes. Esto implica integrar un enlace directo en la plataforma de citas médicas. Cabe resaltar que también tiene la debida intervención de un operador para detener o dar paso a las personas con el flujo debido de las personas.

The screenshot shows the 'Asistente de Citas Médicas' (Medical Appointment Assistant) interface. The header includes a medical icon and the title 'Asistente de Citas Médicas'. Navigation buttons at the top are 'Exámenes', 'Pago de Citas', 'Autorización', 'Lobby', and 'Registrarse'. The main content area is divided into two sections. On the left, under the heading 'Ticket', there is a text input field containing 'GC005'. Below this, under 'Identificador de la cita', there is a text input field containing '3211234' and a button labeled 'Pasar a la Sala de Espera'. At the bottom left, there are buttons for 'Agendar', 'Inicio', and 'Salir'. On the right side, under 'Espacio de', there is a dropdown menu showing 'Samuel Arango Diaz', and buttons for 'Cancelar Cita' and 'Visualizar Citas'. The footer contains copyright information and an email address: 'Email: asistentedecitasmedicas@gmail.com'.

The screenshot shows the 'SALA DE ESPERA ATENCIÓN MÉDICA' (Medical Attention Waiting Room) interface. The header is identical to the previous screen. The main content area features a large table with columns for 'Nombre', 'Apellido(s)', and 'Ticket'. To the right of the table, there is a section titled 'Estado de la sala' with a button labeled 'Flujo Continuo'. On the far right, under 'Espacio de', there is a dropdown menu showing 'Operador 123', a button labeled 'Regresar', and buttons for 'Continuar Flujo' and 'Detener Flujo'. The footer is identical to the previous screen.

8. Pago de citas

Para el proceso de pago de citas, fue esencial diseñar una interfaz intuitiva y con diferentes filtros, así como en la demás sección donde si la contraseña es incorrecta o si no se encuentra la cita buscada para pagar o si el usuario no existe. Asegurando a los usuarios una interacción de forma rápida , intuitiva y práctica y sin complicaciones. Los parámetros solicitados para revisar si esa cita ya se encuentra paga se digitan los datos solicitados por la interfaz y sus debidos parámetros al verificar se despliegan dos posibles rutas 1 de que la cita ya es paga y usted puede ser atendido y la 2 es que esa cita no está paga y usted debe pagarla en una cola de caja que también es monitoreada por el operador.

The screenshot shows the 'Asistente de Citas Médicas' login interface. It features a header with the application name and a navigation bar with buttons for 'Exámenes', 'Lobby', 'Autorización', 'Iniciar Sesión', and 'Registrarse'. The main form area contains input fields for 'Nombre(S):' (filled with 'Samuel'), 'Apellido(S):' (filled with 'Arango Díaz'), 'Ticket:' (filled with 'GA001'), 'Contraseña:' (masked with '***'), 'Identificación:' (filled with '1116774764'), and 'Identificador de la cita:' (filled with '312321'). There is a 'Mostrar Contraseña' toggle and a 'PagoExámenes' button. A footer contains copyright information and an email address.

Asistente de Citas Médicas

Exámenes Lobby Autorización Iniciar Sesión Registrarse

Nombre(S):
Samuel

Apellido(S):
Arango Díaz

Identificación:
1116774764

Identificador de la cita:
312321

Ticket:
GA001

Contraseña:
*** ☐ Mostrar Contraseña

Si desea pagar un examen de click en el siguiente boton y recuerde que debe cumplir con una autorización

PagoExámenes

Verificar

Derechos de autor © 2024 AsistenteDeCitasSA. Todos los derechos reservados. Prohibida la reproducción total o parcial de este sitio web, incluidos textos, imágenes y diseño, sin la autorización previa por escrito de AsistenteDeCitasMedicasSA. Email: asistentedecitasmedicas@gmail.com

The screenshot shows the 'SALA DE ESPERA PAGO DE CITAS' interface. It features a header with the application name and a navigation bar with buttons for 'Exámenes', 'Pago de Citas', 'Autorización', 'Lobby', and 'Registrarse'. The main area contains a table with columns 'Nombre', 'Apellido(s)', and 'Ticket'. To the right of the table is a 'Estado de la sala' section with a 'Detenido' button. On the far right, there is a 'Espacio de Operador 123' section with 'Regresar', 'Continuar Flujo', and 'Detener Flujo' buttons. A footer contains copyright information and an email address.

Asistente de Citas Médicas

Exámenes Pago de Citas Autorización Lobby Registrarse

SALA DE ESPERA PAGO DE CITAS

Nombre	Apellido(s)	Ticket
--------	-------------	--------

Estado de la sala
Detenido

Espacio de Operador 123

Regresar

Continuar Flujo

Detener Flujo


Derechos de autor © 2024 AsistenteDeCitasSA. Todos los derechos reservados. Prohibida la reproducción total o parcial de este sitio web, incluidos textos, imágenes y diseño, sin la autorización previa por escrito de AsistenteDeCitasMedicasSA. Email: asistentedecitasmedicas@gmail.com

9. Autorización/Registro de exámenes

Para el proceso de autorización o registro de exámenes médicos, es importante diseñar la interfaz que permite a los usuarios la solicitud y autorización de la realización de pruebas médicas de forma eficiente cabe resaltar que si es un examen dejado luego de una cita médica el examen se selecciona de forma automática y si es un examen solicitado por el usuario de forma independiente deberá tener mínimamente un control con nosotros para crear una historia clínica e ir de la mano con el marco legal. Bajo la ley de **Ley de 1551 de 2015** que resalta que ningún establecimiento que presta sus servicios a la salud realizará exámenes médicos a personas sin antes saber su correcto estado de salud para su debido examen. Cabe decir que la autorización tiene su sala de espera qué va ligado al sistema LIFO monitoreado por el operador para tener y continuar con el flujo de la sala para los usuarios y su correcta autorización de examen. En la parte de autorizaciones también se puede visualizar todas las autorizaciones a nombre de cierto usuario y cancelarlas si es el caso deseado por el usuario.

The screenshot shows the 'Asistente de Citas Médicas' (Medical Appointment Assistant) interface. The title bar includes a medical icon and the text 'Asistente de Citas Médicas'. The navigation bar contains tabs: 'Exámenes', 'Pago de Citas', 'Lobby', 'Iniciar Sesión', and 'Registrarse'. The main form area is divided into two columns. The left column contains input fields for 'Nombre(S):', 'Apellido(S):', 'Identificación:', and 'Identificador de la cita:'. The right column contains input fields for 'Ticket:', 'Contraseña:', and a 'Mostrar Contraseña' link. Below the password field is a section for 'Tipo de Examen a Autorizar' with radio buttons for 'Sangíneo', 'Cefálico', 'Cráneo', 'Radiografía', 'Orina', and 'Próstata'. At the bottom of the form are buttons for 'Registro' and 'Visualizar Autorizaciones Exitosas'. A note at the bottom states: 'Si deseas visualizar todas las autorizaciones a tu nombre solo completa el campo, nombre, apellido, identificación y contraseña. Recuerda también que el tipo de examen se asigna automáticamente dependiendo del ticket de tu cita.' The footer contains copyright information and an email address: 'Email: asistentedecitasmedicas@gmail.com'.

The screenshot shows the 'SALA DE ESPERA AUTORIZACIONES' (Authorization Waiting Room) interface. The title bar includes a medical icon and the text 'Asistente de Citas Médicas'. The navigation bar contains tabs: 'Exámenes', 'Pago de Citas', 'Autorización', 'Lobby', and 'Registrarse'. The main area is titled 'SALA DE ESPERA AUTORIZACIONES'. On the left, there is a table with columns 'Nombre', 'Apellido(s)', and 'Ticket'. Below the table is a large empty box. On the right, there is a section for 'Espacio de Operador 123' with a 'Regresar' button. Below this are buttons for 'Continuar Flujo' and 'Detener Flujo'. At the bottom, there is a section for 'Estado de la sala' with a 'Flujo Continuo' button. The footer contains copyright information and an email address: 'Email: asistentedecitasmedicas@gmail.com'.



Asistente de Citas Médicas

Exámenes

Pago de Citas

Lobby

Iniciar Sesión

Registrarse

Nombre	Apellido(S)	Iden.T	Iden.Cta	Ticket	Costo	Examen	Iden.Autbri	Info	Autorizada?	TicketAuto
Samuel	Arango Diaz	1116774764	9650	PV001	85000	Cenfálico	41326	Un examen que s...	AutorizacionExit...	CE001

Visualizar


Cancelar Autorización

Volver

Espacio de
Samuel Arango Diaz

Derechos de autor © 2024 AsistenteDeCitasSA. Todos los derechos reservados. Prohibida la reproducción total o parcial de este sitio web, incluidos textos, imágenes y diseño, sin la autorización previa por escrito de AsistenteDeCitasMedicasSA.

Email: asistentedecitasmedicas@gmail.com



Asistente de Citas Médicas

Exámenes

Pago de Citas

Lobby

Iniciar Sesión

Registrarse

Identificador de autorización

Ticket de Cita

Por favor verifique los datos del agendamiento de autorización

Autorizo la cancelación de mi cita

CONFIRMAR CANCELACION

Volver

Espacio de
Samuel Arango Diaz

Derechos de autor © 2024 AsistenteDeCitasSA. Todos los derechos reservados. Prohibida la reproducción total o parcial de este sitio web, incluidos textos, imágenes y diseño, sin la autorización previa por escrito de AsistenteDeCitasMedicasSA.


Email: asistentedecitasmedicas@gmail.com

10. Pago de exámenes/A partir de una autorización exitosa

Sección de exámenes aquí se encuentran 3 diferentes opciones para proceder la primera es proceder con el pago que se pagara el examen a partir de autorización de exámenes es exitosa es decir el registro de los exámenes es exitosa se procede a pagar para en consecuencia pasar a ser atendidos y asegura para los usuarios una experiencia fluida y coherente. También cuenta con una visualización de exámenes a nombre del usuario también hay distintos casos si el usuario digita los datos de un examen ya pago saldrá la notificación de que ese examen ya este pago y que puede proceder a la sala de atención gestionada por el operador. También se cuenta con una cancelación de exámenes ya que examen es como una cita la compañía que preste el servicio entonces se puede cancelar un examen ya pago y no asistido. Pero si ya está asistido no podrá cancelarse y para cancelar un examen no pago que es su debida autorización solamente se deberá cancelar dicha autorización.

The screenshot shows the 'Asistente de Citas Médicas' web application with the 'Lobby' tab selected. The header includes a medical icon and the title 'Asistente de Citas Médicas'. Navigation tabs are 'Lobby', 'Pago de Citas', 'Autorización', 'Iniciar Sesión', and 'Registrarse'. The main form contains input fields for 'Nombre(s)', 'Apellidos', 'N° Identificación', 'Contraseña', 'Identificador de Autorización', and 'Ticket de Autorización'. There is a 'Mostrar Contraseña' link and buttons for 'Agendar/PassarA Pagar', 'Visualizar Exámenes', and 'Sala de Espera'. A note states: 'Si deseas visualizar todos los exámenes a tu nombre solo completa el campo, nombre, apellido, identificación y contraseña'. The footer contains copyright information and an email address.

The screenshot shows the 'Asistente de Citas Médicas' web application with the 'Exámenes' tab selected. The header is the same as the previous screenshot. The main section is titled 'SALA DE ESPERA PAGO DE EXAMENES'. It features a table with columns 'Nombre', 'Apellido(s)', and 'Ticket'. To the right of the table, there is a 'Estado de la sala' section with a 'Flujo Continuo' button. On the far right, there is a 'Espacio de' section with 'Operador 123', a 'Regresar' button, and 'Continuar Flujo' and 'Detener Flujo' buttons. The footer is identical to the previous screenshot.



Asistente de Citas Médicas

Lobby

Pago de Citas

Autorización

Iniciar Sesión

Registrarse

Nombre	Apellido(s)	IdentExamen.	TicketExamen	Costo	TipoExa	Descripción	Pago?	Asistido?
Samuel	Arango Diaz	41326	CE001	85000	Cenfálico	Un examen que se enfo...	examenPago	Asistido

Visualizar


Cancelar Examen

Volver

Espacio de
Samuel Arango Diaz

Derechos de autor © 2024 AsistenteDeCitasSA. Todos los derechos reservados. Prohibida la reproducción total o parcial de este sitio web, incluidos textos, imágenes y diseño, sin la autorización previa por escrito de AsistenteDeCitasMedicasSA.

Email: asistentedecitasmedicas@gmail.com



Asistente de Citas Médicas

Lobby

Pago de Citas

Autorización

Iniciar Sesión

Registrarse

Identificador de examen

213141

Ticket de Examen

GC546

Espacio de
Samuel Arango Diaz

Por favor verifique los datos del agendamiento de autorización

☒

Autorizo la cancelación de mi cita

CONFIRMAR CANCELACION

Volver

Derechos de autor © 2024 AsistenteDeCitasSA. Todos los derechos reservados. Prohibida la reproducción total o parcial de este sitio web, incluidos textos, imágenes y diseño, sin la autorización previa por escrito de AsistenteDeCitasMedicasSA.

Email: asistentedecitasmedicas@gmail.com

11. Atención de exámenes/A partir de un Pago exitoso

Una vez que el pago por los exámenes médicos ha sido exitoso, es crucial asegurarse de que el proceso de atención de los exámenes sea eficiente y satisfactorio para los usuarios. Aquí hay algunas consideraciones importantes para esta parte:

Confirmación de pago: Después de que el pago sea exitoso, proporciona al usuario una confirmación clara de que la transacción se ha completado con éxito. Esto incluye un mensaje en pantalla.

Programación de exámenes: Una vez que se haya confirmado el pago, permite al usuario programar los exámenes médicos conveniente para ellos. Esto puede hacerse a través de la sección de autorizaciones.

The screenshot displays the 'Asistente de Citas Médicas' web application. The header includes a medical icon and the title 'Asistente de Citas Médicas'. A navigation bar contains buttons for 'Exámenes', 'Pago de Citas', 'Autorización', 'Lobby', and 'Registrarse'. The main content area is titled 'SALA DE ESPERA ATENCIÓN DE EXAMENES'. On the left, there is a table with columns 'Nombre', 'Apellido(s)', and 'Ticket'. To the right of the table, there is a section labeled 'Estado de la sala' with a button 'Flujo Continuo'. On the far right, there is a section labeled 'Espacio de' with a dropdown menu showing 'Operador 123', a 'Regresar' button, and two buttons labeled 'Continuar Flujo' and 'Detener Flujo'. The footer contains copyright information and an email address: 'Email: asistentedecitasmedicas@gmail.com'.

El prototipo del diseño inicial lo podrá encontrar adjuntado en el siguiente link <https://www.figma.com/file/z6BQy7Ulay65LyDyWPGAgX/Untitled?type=design&node-id=0%3A1&mode=design&t=V8EQUcQML9dum6BY-1>

que contiene todas las vistas del aplicativo.

9.4.2 Visualización de código

En el desarrollo de las interfaces de usuario para el sistema de gestión de citas médicas, se ha prestado especial atención a la estructura y organización del código subyacente que define estas vistas. Aquí hay una de las consideraciones más importantes que tuvimos en cuenta al momento de desempeñar la elaboración. En la presente sección que recae en el nombre de visualización de código se enfatizará, mostrará y se expondrá una vista de las partes de código interno que dan pie al cumplimiento de las principales partes con base a la lista de requerimientos del proyecto solicitado.

1. **Modularidad y reutilización**, el código que define las vistas se ha estructurado de distintas formas de forma modular, lo que permite la reutilización de componentes en diferentes partes del sistema. Esto facilita la consistencia visual y el funcional en todas las vistas y promueve la eficiencia en el desarrollo.
2. **Separación de responsabilidades**, Se ha aplicado el principio de separación de responsabilidades para garantizar que el código que define las vistas este claramente separado de la lógica de la programación orientada a eventos que son las acciones con los frames y el manejo de datos. Mejorando la legibilidad del código y facilitando la colaboración entre desarrolladores.
3. **Accesibilidad**, Se han seguido las prácticas de accesibilidad garantizando que las vistas y el funcionamiento vayan de la mano. De forma práctica y funcional para todos los usuarios, incluyendo aquellos con discapacidades visuales o motoras.
4. **Patrón MVC**, el sistema de asistente de citas médicas se divide en tres componentes principales uno de ellos es el modelo que representa la estructura de datos y la lógica de negocio del sistema. En la división por paquetes y clases, las clases que representan el modelo suelen encontrarse en paquetes que describen su funcionamiento con la etiqueta de (Gestor) otra de las responsabilidades son las vistas de mostrar la interfaz al usuario. En la división por paquetes y clases, que representan las vistas suelen encontrarse en el paquete vistas. Estas clases definen la presentación de los datos y manejan la interacción con el usuario estos paquetes llevan la etiqueta de (IGU) y finalmente y controlador que actúa como intermediario entre el modelo y la vista con la parametrización de datos respondiendo a las solicitudes del usuario y actualizando la vista o el modelo estos paquetes llevan como etiqueta (Lógica)

Inicialmente el código presentado en la parte inferior implementa la funcionalidad para registrar usuarios en el sistema. El método de registrarBotomActionPerformed, este método se llama cuando se presiona un botón de registrar en la interfaz de usuario. Comienza con un condicional que verifica si los datos ingresados son válidos y se ha seleccionado un botón de opción (En este caso términos y condiciones). Si los datos son válidos se ha seleccionado el botón de opción, se procede a crear un nuevo objeto de la clase Usuario con la información ingresada donde la información solicitada para hacer un registro exitoso es: Nombre, Apellido, Identificación, número de teléfono y el ID asociado al historial clínico es la identificación del usuario.

- Luego, se verifica si el usuario ya existe en el sistema llamando al método “usuarioExiste” de la clase “ValidadorUsuarioExistente”. Si el usuario ya existe, se muestra una ventana informando al usuario sobre la situación al respecto.
- Si el usuario no existe, se intenta guardar el nuevo usuario llamando al método “guardarUsuario” de la clase “ManejadorArchivo”. Si el usuario se guarda adecuadamente, se muestra una ventana de registro exitoso; de lo contrario, se muestra una ventana de registro fallido si existe un tipo de error en el registro de los datos.
- Si los datos no son válidos o si no se ha seleccionado el botón correspondiente de opción, se muestra una ventana de registro fallido. Posibles casos que en número de teléfono y edad el usuario ponga palabras del contrario de números enteros como es habitual llenar esta información.

```

private void registrarbotomActionPerformed(java.awt.event.ActionEvent evt) {
    if (validarDatos() && jRadioButton1.isSelected()) //Condicional
    {
        //Creamos el usuario
        Usuario usuario = new Usuario(nombre: nombreEntrada.getText(),
            apellido: apellidoEntrada1.getText(),
            identificacion: identificacionEntrada1.getText(),
            edad: edadEntrada1.getText(),
            telefono: telefonodoEntrada.getText(),
            new String(value: passwordEntrada1.getPassword()));

        if(ValidadorUsuarioExistente.usuarioExiste(usuario))
        {
            this.dispose();
            UsuarioYaExistente ventana= new UsuarioYaExistente();
            ventana.setVisible(b: true);
            ventana.setLocationRelativeTo(c: null);
            ventana.setResizable(resizable: false);
        }
        else
        {
            boolean usuarioGuardado = ManejadorArchivo.guardarUsuario(usuario);
            if(usuarioGuardado)
            {
                this.dispose();
                RegistroExitoso regie=new RegistroExitoso(usuario);
                regie.setVisible(b: true);
                regie.setLocationRelativeTo(c: null);
                regie.setResizable(resizable: false);
            }
            else
            {
                this.dispose();
                RegistroFallido fallido = new RegistroFallido(usuario);
            }
        }
    }
}

```

```

        fallido.setVisible(b: true);
        fallido.setLocationRelativeTo(c: null);
        fallido.setResizable(resizable: false);
    }
}
else
{
    Usuario usuario = new Usuario(nombre: nombreEntrada.getText(),
    apellido: apellidoEntrada1.getText(),
    identificacion: identificacionEntrada1.getText(),
    edad: edadEntrada1.getText() ,
    telefono: telefonodoEntrada.getText(),
    new String(value: passwordEntrada1.getPassword()));

    this.dispose();//Mostramos la ventanade registro fallido si los datos
    //no son validos
    RegistroFallido fallido = new RegistroFallido(usuario);
    fallido.setVisible(b: true);
    fallido.setLocationRelativeTo(c: null);
    fallido.setResizable(resizable: false);
}
}

```


Método usuarioExiste. Este método relevante en el funcionamiento adecuado del primer requerimiento se encuentra en la clase “ValidadorUsuarioExistente” y se utiliza para verificar si un usuario ya existe en el sistema. Leyendo un archivo de texto que contiene información de usuario utilizando un “BufferedReader”. Luego, compara la identificación del usuario proporcionado con las identificaciones de los usuarios almacenados en el archivo. Si se encuentra una coincidencia, devuelve “True”; de lo contrario, devuelve “False”.

```
private boolean usuarioExisteEnArchivo(Usuario usuario) {  
    try (BufferedReader reader = new BufferedReader(new FileReader(fileName: RUTA_PRUEBA))) {  
        String linea;  
        while ((linea = reader.readLine()) != null) {  
            Usuario usuarioGuardado = parsearUsuario(linea);  
            if (usuarioGuardado != null && usuario.getIdentificacion().equals(anObject: usuarioGuardado.getIdentificacion()))  
                return true; // El usuario se encuentra en el archivo  
        }  
    }  
    catch (IOException e) {  
        e.printStackTrace();  
    }  
    return false; // El usuario no se encuentra en el archivo  
}
```

La sección de Método guardarUsuario. Este método está en la clase “ManejadorArchivo” y se utiliza para guardar un objeto de la clase “Usuario” en un archivo de texto. Utilizando un “BufferedWriter” para escribir la información del usuario en el archivo. Si se produce un error durante el proceso de la escritura de la información, se maneja la excepción y devuelve “false”; de lo contrario se devuelve “true” para indicar que el usuario se guardó correctamente.

```
~/  
public static boolean guardarUsuario(Usuario usuario) {  
    try (BufferedWriter writer = new BufferedWriter(new FileWriter(fileName: RUTA_ARCHIVO, append:true))) {  
        String usuarioStr = formatUsuario(usuario); // Formatear el usuario como una cadena  
        writer.write(str:usuarioStr);  
        writer.newLine();  
        return true; // Indicar que el usuario se guardó correctamente  
    } catch (IOException e) {  
        e.printStackTrace(); // Manejo de la excepción: imprimir la traza en caso de error  
        return false; // Indicar que hubo un error al guardar el usuario  
    }  
}
```

Aquí se encuentra el sistema de gestión de citas, el cual recibe como insumo la información de especialidad, profesional y motivo de cita. Todo inicia en el método ConfirmarCitaActionPerformed, este método se activa cuando se presiona un botón para confirmar una cita médica en la interfaz de usuario. Comienza verificando si se ha seleccionado algún médico para la cita. Luego, convierte los datos ingresados en los campos de texto a los formatos adecuados (enteros, cadenas, booleanos) para su procedimiento adecuado.

- Si se ha seleccionado un médico, se procede a determinar la especialidad y el motivo de la cita seleccionados por el usuario. Además, se selecciona el médico elegido. Luego, se creará un objeto “confirmacionDelAgendamientoDeLaCita” con los datos pertinentes de la cita y se mostrará en una ventana de confirmación.
- Si no se ha seleccionado ningún médico, no se realiza ninguna acción adicional.

```
private void ConfirmarCitaActionPerformed(java.awt.event.ActionEvent evt) {  
    boolean doctorSeleccionado = doctorArango.isSelected() ||  
        doctorJeison.isSelected() || doctorJuanCarlos.isSelected();  
  
    //Convertir el texto de identificador de String a int  
    int identificador = Integer.parseInt(s: identificadorDeCita.getText());  
  
    // Convertir el texto de Ticket a String  
    String ticket = Ticket.getText();  
  
    // Convertir el texto de Costo a entero  
    int costo1 = Integer.parseInt(s: Costo.getText());  
    boolean pagarSeleccionado=PagarBoton.isSelected();  
  
    if (doctorSeleccionado)  
    {  
        //Aplicacion de logica similar a la de especialidad y motivo la diferencia  
  
        String especialidadSeleccionadaTexto = "";  
        String motivoSeleccionadoTexto="";  
        String doctorSeleccionado1 = "";  
  
        //DEPENDIENDO DE LAS OPCIONES QUE SE ESCOJAN SE AGREGA EL TEXTO PARA GUARDAR AL RESPECO  
        if (boton1.isSelected()) especialidadSeleccionadaTexto = boton1.getText();  
        else if (boton2.isSelected()) especialidadSeleccionadaTexto = boton2.getText();  
        else if (boton3.isSelected()) especialidadSeleccionadaTexto = boton3.getText();  
        else if (boton4.isSelected()) especialidadSeleccionadaTexto = boton4.getText();  
        else if (boton5.isSelected()) especialidadSeleccionadaTexto = boton5.getText();  
    }  
}
```

```

// Verificamos el motivo seleccionado
if (motivo1.isSelected()) motivoSeleccionadoTexto = motivo1.getText();
else if (motivo2.isSelected()) motivoSeleccionadoTexto = motivo2.getText();
else if (motivo3.isSelected()) motivoSeleccionadoTexto = motivo3.getText();

if (doctorArango.isSelected()) doctorSeleccionado1= doctorArango.getText();
else if (doctorJeison.isSelected()) doctorSeleccionado1 = doctorJeison.getText();
else if (doctorJuanCarlos.isSelected()) doctorSeleccionado1= doctorJuanCarlos.getText();

// Pasar los valores generados al constructor de
//confirmacionDelAgendamientoDeLaCita
confirmacionDelAgendamientoDeLaCita confir =
new confirmacionDelAgendamientoDeLaCita(
    nombreUsuario, apellidoUsuario: apellidoUsaurio, identificacion,
    especialidadMotivoSeleccionados: especialidadSeleccionadaTexto,
    identificador, ticket, costol,
    pagarSeleccionado, doctorSeleccionado: doctorSeleccionado1, motivoSeleccionadoTexto
);

this.dispose();
confir.setVisible(b: true);
confir.setLocationRelativeTo(c: null);
confir.setResizable(resizable:false);
}

else
{
    //No uso nada en consola ya que se tiene en cuenta que el usuario
    //usara el programa desde la interfaz y el inicio sera en un .exe
}
}

```

Método generarIdentificadorDeCita, este método genera un identificador único para la cita médica de forma aleatoria. Utiliza el método “random()” de la clase “Math” para generar un numero aleatorio entre 1 y 50000.

Método generar Ticket. Este método genera un numero de ticket a partir del ticket a crear dependiendo de su especialidad y motivo teniendo en cuenta los ya almacenados dando un ticket incremental conforme a las citas que ya existan en una sección específica. Combinando la identificación de la especialidad y el motivo de la cita con un número incremental con base a lo almacenado para crear un ticket único y practico de identificar.

```
//Generamos un identificador de forma aleatoria con el metodo random
private int generarIdentificadorDeCita() { //perteneciendo a la clase Math
    return (int) (Math.random()*50000)+1; //interna que se importo
}

private String generarTicket() {
    StringBuilder ticket = new StringBuilder();

    // Agregar la identificación de especialidad y motivo (por ejemplo, GV001)
    ticket.append(str: obtenerIdentificacionEspecialidadMotivo());

    // Agregar un número aleatorio como secuencia
    ticket.append(str: obtenerSiguienteNumeroCita());
    return ticket.toString();
}

private String obtenerIdentificacionEspecialidadMotivo() {
    StringBuilder identificacion = new StringBuilder();
    if (boton1.isSelected()) {
        identificacion.append(str: "G"); // Ejemplo de código para medicina general
    } else if (boton2.isSelected()) {
        identificacion.append(str: "P"); // Ejemplo de código para pediatría
    } else if (boton3.isSelected()) {
        identificacion.append(str: "GI"); // Ejemplo de código para ginecología
    } else if (boton4.isSelected()) {
        identificacion.append(str: "C"); // Ejemplo de código para cardiología
    } else if (boton5.isSelected()) {
        identificacion.append(str: "D"); // Ejemplo de código para dermatología
    }
}
```

```

if (motivo1.isSelected()) {
    identificacion.append(str: "V"); // Ejemplo de código para valoración
} else if (motivo2.isSelected()) {
    identificacion.append(str: "A"); // Ejemplo de código para asesoramien
} else if (motivo3.isSelected()) {
    identificacion.append(str: "C"); // Ejemplo de código para control
}

return identificacion.toString();

}

/**
 * Obtenemos el numero de cita para la identificacion especifica, El # d
 * cita se basa en el contenido del archivo de citas
 * @return
 */
private String obtenerSiguienteNumeroCita() {
//Obtenemos la identificacion de la cita para la cual se necesita
String identificacion = obtenerIdentificacionEspecialidadMotivo();
String siguienteNumeroCita = "";

```

Método obtenerIdentificacionEspecialidadMotivo. Este método combina la identificación de la especialidad y el motivo seleccionados por el usuario para generar una identificación única para la cita médica.

Método obtenerSiguieteNumeroCita. Este método determina el siguiente número de cita disponible para una identificación específica (basada en la especialidad y el motivo). Lee un archivo de citas existentes y cuenta el número de citas asociadas con la identificación actual. Luego, devuelve el siguiente numero de la cita disponible según este contador.

Selección del doctor. Antes de confirmar la cita, el código verifica si se ha seleccionado algún doctor para la cita médica. Esta selección se realiza mediante los botones selección “doctor, Arango, Jeison, JuanCarlos”. Estos botones están asociados a la acción de seleccionar el médico correspondiente.

- Si al menos uno de los botones de selección de doctor está marcado, se procede con la confirmación de la cita. El código determina cuál de los botones está seleccionado y guarda el nombre del médico seleccionado en la variable “doctorSeleccionado1”. Esta información se utilizará más adelante al confirmar la cita médica.
- Si ninguno de los botones de selección de doctor está marcado, el código no realiza ninguna acción adicional, ya que se asume que el usuario no ha completado la selección del médico y, por lo tanto, no se puede confirmar la cita sin esta información.

```

private String obtenerSiguienteNumeroCita() {
    //Obtenemos la identificacion de la cita para la cual se necesita
    String identificacion = obtenerIdentificacionEspecialidadMotivo();
    String siguienteNumeroCita = "";

    try {
        //Abrimos el archivo de citas para leer
        FileReader fr = new FileReader(
fileName: "src/main/java/com/asistentemedico/asistentedecitas/persistencia/citas.txt");
        BufferedReader br = new BufferedReader(in: fr);

        //necesitamos un contador para revisar las citas existentes
        int contador = 0;

        // Leer cada línea del archivo y contar las citas existentes
        //para la identificación actual
        String linea;
        while ((linea = br.readLine()) != null) {
            //Si la línea contiene la identificación, incrementaremos de 1 en 1
            //respecto a lo solicitado y con base a las letras especialidad,
            //motivo
            if (linea.contains(s: identificacion))
            {
                contador++;
            }
        }

        // Construir el siguiente número de cita cumpliendo con el requerimiento
        siguienteNumeroCita = String.format(format:"%03d", contador + 1);
        //Construir el siguiente numero de cita (Se suma 1 al contador y se for
        br.close(); //matea
        fr.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

```



```

        if (motivo1.isSelected()) motivoSeleccionadoTexto = motivo1.getText();
        else if (motivo2.isSelected()) motivoSeleccionadoTexto = motivo2.getText();
        else if (motivo3.isSelected()) motivoSeleccionadoTexto = motivo3.getText();

        if (especialidadSeleccionada && motivoSeleccionado && !motivoExamenSeleccionado) { //Especialidad
            //evaluamos obteniendo su respectivo texto para pasar al siguiente

            // Si se cumple, proceder al AgendamientoDeCitas
            AgendamientoDeCitas agend = new AgendamientoDeCitas(nombreUsuario,
            apellidoUsaurio: apellidoUsuario, identificacion,
            especialidadMotivoSeleccionados: especialidadSeleccionadaTexto, motivoSeleccionado: motivoSeleccionadoTexto );
            //le pasamos los parametros Nom,Ape,Iden,especialidad
            this.dispose();
            agend.setVisible(b: true);
            agend.setLocationRelativeTo(c: null);
            agend.setResizable(resizable: false);
        }
        else if(motivoExamenSeleccionado){
            this.dispose();
            EntradadeExamenenes entr = new EntradadeExamenenes();
            entr.setVisible(b: true);
            entr.setLocationRelativeTo(c: null);
            entr.setResizable(resizable: false);
        }

        else {
            ErrorAlAgendarOPasarALaSalaDeEspera agend = //Si el usuario no marco
            new ErrorAlAgendarOPasarALaSalaDeEspera(nombreUsuario, //correctamente
            apellidoUsuario, identificacion); //al agendar se le volvera a permitir
            //coregir su error
            this.dispose();
            agend.setVisible(b: true);
            agend.setLocationRelativeTo(c: null);
            agend.setResizable(resizable: false);
        }
    }

```

Luego de ese filtrado de información viene el guardado de la cita que recae en el Frame “confirmacionDelAgendamientoDeLaCita”:

Donde el método AgendarActionPerformed. Este método se activa cuando el usuario confirma la cita médica. Comienza obteniendo los datos necesarios para la cita a partir de los campos de entrada en la interfaz de usuario. Estos datos incluyen el nombre y apellido del usuario, su identificación, el identificador de la cita, el ticket de la cita, el costo de la cita, el nombre del doctor seleccionado, si la cita está por defecto como control dirá que es “PAGA” de lo contrario inicialmente el flag será “NOPAGA”. Además, se inicializa el estado de asistencia como “No asistida”

- Luego, ensambla una cadena “cita” que contiene todos estos datos separados por comas. Esta cadena se utiliza para manejar y almacenar temporalmente la información de la cita antes de guardarla en un archivo.
- Después, invoca el método “guardarCita” del “GestorCitas” para guardar la cita en un archivo. Este método simplemente describe la cadena de la cita en un archivo de texto.
- Y se crea una instancia del marco de agendamiento exitoso “agendamientoExitoso” para información para el usuario que la cita se ha agendado correctamente. Esta ventana muestra algunos detalles de la cita confirmada y proporciona retroalimentación al usuario.

```

private void AgendarActionPerformed(java.awt.event.ActionEvent evt) {
    // Obtener los datos de la cita
    String nombre = nombreUsuario;
    String apellido = apellidoUsuario;
    String identificacionUsuario = identificacion;
    int identificadorCita = Integer.parseInt(s: identificadorDeCita.getText());
    String ticketCita = Ticket.getText();
    int costoCita = Integer.parseInt(s: Costo1.getText());
    String doctor = obtenerDoctorSeleccionado();
    boolean citaPaga = PagarBoton.isSelected();
    String especialidadMotivo = obtenerEspecialidadMotivo();
    String motivo = obtenerMotivo();
    String asistencia="No asistida";
    //EVALUAMOS PARA ALMACENAR PROXIMAMENTE EN EL METODO DE GUARDAR CITA
    String cita = nombre + ", " + apellido + ", " + identificacionUsuario +
    ", " + identificadorCita + ", " + ticketCita + ", " + costoCita +
    ", " + doctor + ", " + (citaPaga ? "Paga" : "No Paga")
    + ", " + especialidadMotivo + ", " + asistencia + ", " + motivo ;
    //Una vez que el usuario ya hubiera ravisa todo los datos pues finalmente
    //guardamos la cita con base a lo que se visualiza inicialmente, la asistencia
    //sera no asistida y preguntamos si paga o no paga para posteriormente guardar
    //ya que los controles se guardan como pagos ya que son gratuitos
    GestorCitas.guardarCita(cita);
    //frame de agendamiento exitoso
    agendamientoExitoso inic = new agendamientoExitoso(nombreUsuario:nombre,
        apellidoUsuario: apellido, identificacion: identificacionUsuario,
        especialidadMotivoSeleccionados: especialidadMotivo,
        identificador:identificadorCita, ticket:ticketCita, costo1:costoCita,
        pagarSeleccionado: citaPaga, doctorSeleccionado: doctor, motivo);

    inic.setVisible(b: true);
    inic.setLocationRelativeTo(c: null);
    inic.setResizable(resizable:false);
    this.dispose();
}

```

Método guardarCita. Este método, en la clase “GestorCitas”, se utiliza para guardar la cita en un archivo de texto. Utiliza un “FileWriter” y un “PrintWriter” para escribir la cadena de la cita en el archivo especificado. Si ocurre algún error durante el proceso de escritura, se maneja la excepción imprimiendo la traza de la excepción.

Cabe resaltar que inicialmente es una cita de examen el código automáticamente redirecciona al usuario al espacio exclusivamente de exámenes.

```
public static void guardarCita(String cita) {  
  
    try (FileWriter fw = new FileWriter(fileName: RUTA_ARCHIVO,  
        PrintWriter pw = new PrintWriter(out: fw)) {  
        pw.println(x: cita);  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
}
```

Ahora dando pie a la continuidad del pagoDeCitas ya que como se mencionó anteriormente todas las citas inicialmente al almacenarse son citas NoPagas a excepción de los controles su funcionalidad inicialmente es la siguiente:

Método verificarPagoActionPerformed. Este método se activa cuando el usuario realiza una acción, probablemente al intentar verificar si una cita está pagada o no. Comienza verificando si todos los campos requeridos están completos.

- Si todos los campos están completos, se procede a obtener los datos necesarios para la verificación de pago. Esto incluye el nombre y apellido del usuario, su identificación, contraseña, identificador y ticket de la cita.
- Luego, se llama al método “cargarUsuarios” del “ControladorUsuario” para cargar la información de los usuarios desde un archivo. A continuación, se valida el inicio de sesión del usuario llamando al método “validarInicioSesion” del “ControladorUsuario”.
- Si el inicio de sesión es exitoso, se verifica si la cita está pagada llamando al método “verificarCitaPagada” del “GestorCitas”. Dependiendo del resultado de esta verificación, se muestra una ventana informando al usuario si la cita está pagada o no.
- Si ocurre algún error durante el proceso de verificación de pago, se muestra una ventana de error.
- Si no todos los campos están completos, se muestra una ventana de error indicando que algo salió mal al leer los datos.

```
private void VerificarPagoActionPerformed(java.awt.event.ActionEvent evt) {  
    if(camposCompleto())  
    {  
        String nombreUsuario = nombreEntrada.getText();  
        String apellidoUsuario = ApellidoEntrada.getText();  
        String identificacion = identificacionEntrada.getText();  
        String contrasenia = new String(value: contraseñaEntrada.getPassword());  
        String identificadorCita = IdentificadorDeCitaEntrada.getText();  
        String ticket = ticketEntrada.getText();  
  
        ControladorUsuario.cargarUsuarios();  
        if(ControladorUsuario.validarInicioSesion(nombre: nombreUsuario,  
            apellido: apellidoUsuario, identificacion, contrasenia))  
        {  
            boolean citaPaga=GestorCitas.verificarCitaPagada(identificadorCita,  
                ticket);  
            if(citaPaga==true)  
            {  
                this.dispose();  
                CitaPagada silla = new CitaPagada(ticket );  
                silla.setVisible(b: true);  
                silla.setLocationRelativeTo(c: null);  
                silla.setResizable(resizable: false);  
            }  
            else{  
                this.dispose();  
                CitaNoPagada error = new CitaNoPagada(nombreUsuario,  
                    apellidoUsuario,identificacion,  
                    ticket,identificadorDeCita:identificadorCita);  
                error.setVisible(b: true);  
                error.setLocationRelativeTo(c: null);  
                error.setResizable(resizable: false);  
            }  
        }  
    }  
}
```

```
        error.setResizable(resizable: false),
    }
}
else{
    this.dispose();
    ErrorSiPasaAlgoAlLeerDatos error = new ErrorSiPasaAlgoAlLeerDatos();
    error.setVisible(b: true);
    error.setLocationRelativeTo(c: null);
    error.setResizable(resizable: false);
}
}
else
{
    this.dispose();
    ErrorSiPasaAlgoAlLeerDatos error = new ErrorSiPasaAlgoAlLeerDatos();
    error.setVisible(b: true);
    error.setLocationRelativeTo(c: null);
    error.setResizable(resizable: false);
}
```

Método cargarUsuarios. Este método, en el “ControladorUsuario”, se utiliza para cargar la información de los usuarios desde un archivo de texto. Utiliza un “BufferedReader” para leer el archivo línea por línea y, para cada línea, llama al método “parsearUsuario” para convertir la línea en un objeto “Usuario” y lo agrega a la lista de usuarios.

```
public static void cargarUsuarios() {  
    try (BufferedReader reader = new BufferedReader(new FileReader(fileName: RUTA_ARCHIVO))) {  
        String linea;  
        while ((linea = reader.readLine()) != null) {  
            Usuario usuario = parsearUsuario(linea);  
            if (usuario != null) {  
                agregarUsuario(usuario);  
            }  
        }  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
}
```


Método verificarCitaPagada. Este método, en el “GestorCitas”, se utiliza para verificar si una cita está pagada o no. Carga todas las citas desde un archivo utilizando el método “cargarCitasDesdeArchivo”, luego recorre la lista de citas y compara el identificador de la cita y el ticket con los proporcionados por el usuario. Si encuentra una cita que coincide y su estado es “Paga”, devuelve true de lo contrario devuelve “false”.

```
public static boolean verificarCitaPagada(String identificadorCita, String ticket){

    DoblementeEnlazada<Cita> citas = cargarCitasDesdeArchivo(rutaArchivo: RUTA_ARCHIVO);
    Nodo<Cita> nodoActual = citas.getCabeza();

    while(nodoActual != null){

        Cita citaActual = nodoActual.getValor();

        if(citaActual.getIdentificador().equals(anObject: identificadorCita) &&
            citaActual.getTicket().equals(anObject: ticket) &&
            citaActual.getEstado().equalsIgnoreCase(anotherString: "Paga")){
            return true;
        }

        nodoActual = nodoActual.getSiguiente();
    }

    return false;
}
```

Método cargarCitasDesdeArchivo. Este método recibe como parámetro la ruta del archivo que contiene la información de las citas. Utiliza un “FileReader” y un “BufferedReader” para leer el archivo línea por línea. Por cada línea leída, se extraen los campos de la cita utilizando el método “obtenerCampo”. Luego se crea un objeto “Cita” con estos campos y se agrega a la lista doblementeEnlazada.

```
public static DoblementeEnlazada<Cita> cargarCitasDesdeArchivo(String rutaArchivo) {  
    DoblementeEnlazada<Cita> citas = new DoblementeEnlazada<>();  
  
    try (FileReader fr = new FileReader(fileName: rutaArchivo);  
        BufferedReader br = new BufferedReader(in: fr)) {  
        String linea;  
  
        while ((linea = br.readLine()) != null) {  
            String nombre = obtenerCampo(linea, indice: 0);  
            String apellido = obtenerCampo(linea, indice: 1);  
            String identificacion = obtenerCampo(linea, indice: 2);  
            String identificador = obtenerCampo(linea, indice: 3);  
            String ticket = obtenerCampo(linea, indice: 4);  
            String costo = obtenerCampo(linea, indice: 5);  
            String doctor = obtenerCampo(linea, indice: 6);  
            String estado = obtenerCampo(linea, indice: 7);  
            String especialidad = obtenerCampo(linea, indice: 8);  
            String asistencia = obtenerCampo(linea, indice: 9);  
            String motivo = obtenerCampo(linea, indice: 10);  
  
            Cita cita = new Cita(nombre, apellido, identificacion, identificador,  
                                ticket, costo, doctor, estado, especialidad, asistencia, motivo);  
            citas.agregarAlFinal(valor: cita);  
        }  
  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
    return citas;  
}
```

Método obtenerCampo. Este método recibe una línea de texto y un índice y devuelve el campo correspondiente en esta línea. Utiliza una lógica de búsqueda para encontrar el inicio y el final del campo basado en comas. Luego, utiliza el método “substring” para extraer el campo de la línea.

```
// Obtener campo de una línea
private static String obtenerCampo(String linea, int indice){
    int inicio = 0;
    int contador = 0;
    while(contador < indice){
        inicio = linea.indexOf(str: ",", " ", fromIndex: inicio) + 2;
        contador++;
    }
    int fin = linea.indexOf(str: ",", " ", fromIndex: inicio);
    if(fin == -1){
        fin = linea.length();
    }
    return linea.substring(beginIndex: inicio, endIndex: fin).trim();
}
```

Clase DoblementeEnlazada. Esta clase implementa una lista doblemente Enlazada genérica. Tiene un nodo interno que almacena un valor genérico y referencia al nodo siguiente y anterior. Ofrece métodos para agregar elementos al inicio y al final de la lista, eliminar elementos del inicio y del final, imprimir la lista y verificar si está vacía. También proporciona iteradores para recorrer la lista.

```
public class DoblementeEnlazada <T> implements ListaDoblementeEnlazada<T>{
    private Nodo<T> cabeza;
    private Nodo<T> cola;
    private int tamano;

    public DoblementeEnlazada() {
        this.cabeza = null;
        this.colas = null;
        this.tamano = 0;
    }
}
```

```
public static class Nodo<T>{ //Define la clase estatica inter
    T valor;//se declara la variable con nombre de valor
    Nodo<T> siguiente;//Se declara la variable del siguiente
    Nodo<T> anterior;
    public Nodo(T valor) {
        this(valor, null, null);//Este constructor es para el inicio
        //de nodos
    }
    public Nodo(T valor, Nodo<T> siguiente, Nodo<T> anterior) {
        this.valor = valor;//donde sirve para asignar el valor
        this.siguiente = siguiente;//entre el parametro siguiente
        this.anterior = anterior;
    }
    public T getValor() { //Obtiene el valor de ese generico
        return valor;//y lo devuelve, no existe un parametro
    }
    public Nodo<T> getSiguiente() { //Obtiene el siguiente nodo
        return siguiente;//y retornar el siguiente nodo
    }
    public void setSiguiente(Nodo<T> siguiente) { //Establece el siguiente
        this.siguiente = siguiente;//y retorna su valor
    }
    public Nodo<T> getAnterior() { //Misma funcionalidad que get
        return anterior;//anterior al nodo actual de la lista
    }
    public void setAnterior(Nodo<T> anterior) { //Misma funcionalidad
        this.anterior = anterior;
    }
}
```

Clase Cita. Esta clase define una cita médica con varios atributos como nombre, apellido, identificación, identificador de cita, ticket, costo, doctor, estado, especialidad, asistencia y motivo. También proporciona métodos getter y setter para acceder y modificar estos atributos, y un método “toString” para representar la cita como cadena de texto

```
public class Cita {

    private String nombre;
    private String apellido;
    private String identificacion;
    private String identificador;
    private String ticket;
    private String costo;
    private String doctor;
    private String estado;
    private String especialidad;
    private String asistencia;
    private String motivo;

    public Cita(String nombre, String apellido, String identificacion,
String identificador, String ticket, String costo, String doctor,
String estado, String especialidad, String asistencia, String motivo) {

        this.nombre = nombre;
        this.apellido = apellido;
        this.identificacion = identificacion;
        this.identificador = identificador;
        this.ticket = ticket;
        this.costo = costo;
        this.doctor = doctor;
        this.estado = estado;
        this.especialidad = especialidad;
        this.asistencia = asistencia;
        this.motivo = motivo;
    }
}
```

Luego de verificar si la cita no ha sido paga se procede al bloque de pago de citas donde el usuario debe en marcar el botón de pagar para proceder y pasar a la cola de espera de pago de citas donde se visualiza una sala de espera donde están todos los usuarios haciendo fila para pagar la cita.

En esta sección pasarALaSalaDeEspera. Este método se activa cuando se presiona el botón para pasar a la sala de espera para el pago de una cita. Primero, verifica si se ha seleccionado la opción de pago (`PagarBoton.isSelected()`).

Obtención del ticket: Se obtiene el número de ticket de la entrada de texto `ticketEntrada`.

Verificación de la presencia en la sala de espera: Llama al método `usuarioEnSala` del `GestorCitas` para verificar si el usuario ya está en la sala de espera para el pago. Si el usuario ya está en la sala de espera, se muestra la ventana correspondiente (`ColaParaElPagoDeCita`). Si no está en la sala de espera, se guarda en ella mediante el método `guardarEnSalaDePagos` del `GestorCitas`, y luego se muestra la ventana de la sala de espera para el pago.

Cierre de la ventana actual y visualización de la sala de espera: Finalmente, se cierra la ventana actual (`this.dispose()`) y se muestra la ventana de la sala de espera para el pago (`ColaParaElPagoDeCita`).

Manejo del caso en que no se ha seleccionado la opción de pago: En el caso en que no se haya seleccionado la opción de pago, este fragmento de código está vacío (`else{}`). No hay ninguna acción definida para esta situación en este momento.

```
private void pasarALaSalaDeEsperaActionPerformed(java.awt.event.ActionEvent evt) {  
    if(PagarBoton.isSelected()){  
        String ticket1 = ticketEntrada.getText();  
  
        boolean estaEnSala =  
        GestorCitas.usuarioEnSala(nombre: nombreUsuario, apellido: apellidoUsuario, identifiacion: identificacion, ticket:  
  
        if(estaEnSala){  
            this.dispose();  
            ColaParaElPagoDeCita cit = new ColaParaElPagoDeCita(identificadordeautorizacion: ticket1);  
            cit.setVisible(b: true);  
            cit.setLocationRelativeTo(c: null);  
            cit.setResizable(resizable: false);  
        }else{  
            GestorCitas.guardarEnSalaDePagos(nombre: nombreUsuario, apellido: apellidoUsuario, identificacion, ticket: ti  
            this.dispose();  
            ColaParaElPagoDeCita cit = new ColaParaElPagoDeCita(identificadordeautorizacion: ticket1);  
            cit.setVisible(b: true);  
            cit.setLocationRelativeTo(c: null);  
            cit.setResizable(resizable: false);  
        }  
  
        }else{  
  
        }  
    }  
}
```

Método usuarioEnSala. Este método verifica si un usuario con un ticket de cita específico está en la sala de pagos. Utiliza el método cargarCitasParaPagoDesdeArchivo para obtener la lista de citas desde el archivo. Recorre la lista de citas y compara los campos de cada cita con los proporcionados como parámetros (nombre, apellido, identificación y ticket). Si encuentra una coincidencia, devuelve true; de lo contrario, devuelve false. Todo con la correcta parametrización de datos en la clase de SalaDePagos y en ordenamiento con una listadoblementeEnlazada.

```
public static boolean usuarioEnSala(String nombre, String apellido, String
identifiacion, String ticket){

    DoblementeEnlazada<SalaDePago> citas = cargarCitasParaPagoDesdeArchivo(rutaArchivo: RUTA_SALA_DE_PAGOS);
    Nodo<SalaDePago> nodoActual = citas.getCabeza();

    while(nodoActual != null){
        SalaDePago citaActual = nodoActual.getValor();

        if(citaActual.getNombre().equals(anObject: nombre) && citaActual.getApellido().equals(anObject: apellido) &&
            citaActual.getIdentificacion().equals(anObject: identifiacion) && citaActual.getTicket().equals(anObject: ticket)){
            return true;
        }
        nodoActual = nodoActual.getSiguiente();
    }
    return false;
}
```


Método cargarCitasParaPagoDesdeArchivo. Este método carga las citas para el pago desde un archivo y las almacena en una lista doblemente enlazada. Utiliza un objeto `BufferedReader` para leer el archivo línea por línea. En cada línea, se extraen los campos de la cita (nombre, apellido, identificación y ticket) utilizando el método `obtenerCampo`. Se crea un objeto `SalaDePago` con estos campos y se agrega a la lista doblemente enlazada.

Finalmente, se devuelve la lista de citas.

```
public static DoblementeEnlazada<SalaDePago> cargarCitasParaPagoDesdeArchivo(String rutaArchivo){

    DoblementeEnlazada<SalaDePago> citas = new DoblementeEnlazada<>();

    try(FileReader fr = new FileReader(fileName: rutaArchivo);
        BufferedReader br = new BufferedReader(in: fr)){
        String linea;

        while((linea = br.readLine()) != null){
            String nombre = obtenerCampo(linea, indice:0);
            String apellido = obtenerCampo(linea, indice:1);
            String identificacion = obtenerCampo(linea, indice:2);
            String ticket = obtenerCampo(linea, indice:3);
            SalaDePago saladepago = new SalaDePago(nombre, apellido, identificacion, ticket);
            citas.agregarAlFinal(valor: saladepago);
        }
    }catch(IOException e){
        e.printStackTrace();
    }
    return citas;
}
```

Clase SalaDePago. Esta clase modela una cita en la sala de pagos. Tiene cuatro atributos: nombre, apellido, identificación y ticket, que representan los datos del paciente y el identificador de la cita. El constructor SalaDePago inicializa estos atributos con los valores proporcionados. Hay getters y setters para cada atributo para acceder y modificar los datos de la cita. El método toString devuelve una representación en formato de cadena de la cita.

```
public class SalaDePago {

    private String nombre;
    private String apellido;
    private String identificacion;
    private String ticket;

    public SalaDePago(String nombre, String apellido, String identificacion, String ticket) {
        this.nombre = nombre;
        this.apellido = apellido;
        this.identificacion = identificacion;
        this.ticket = ticket;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getApellido() {
        return apellido;
    }

    public void setApellido(String apellido) {
        this.apellido = apellido;
    }

    public String getIdentificacion() {
        return identificacion;
    }
}
```

Método moverPrimerPago de la clase ColaParaElPagoDeCitas. En esta sección se obtiene la primeraCita que se encuentre en la sala para pasarlo a ser atendido con base al método de obtenerPrimeraCitaDesdeArchivo. Si la primera cita es diferente de nulo se operará marcando la cita como pagada ya que el operador tendrá vía libre para atender al cliente que va a pagar las citas y puede detener este flujo cuando el desee o mientras este atendiendo una vez paga la cita automáticamente se elimina la información de la cita que se encuentra en la sala para confirmar el correcto flujo del pago de citas.

```
private void moverPrimerPago() {
    SalaParaElPago primeraCita = obtenerPrimeraCitaDesdeArchivo(rutaArchivo: RUTA_SALA_DE_PAGOS);
    if (primeraCita != null) {
        String ticket = primeraCita.getTicketCita();

        GestorCitas.marcarCitaComoPagada(ticket);

        GestorCitas.eliminarAutorizacionDeSala();
    }
}
```

Método obtenerPrimeraCitaDesdeArchivo. Este método funciona con la parametrización de datos con base a la clase de SalaParaElPago juntando la información obtenida por medio del método de obtenerCampo que es el mismo método ya mencionado y mostrado anteriormente. Cargando así la información en la instancia para marcar correctamente si la cita ha sido paga o no y volviendo a grabar en el archivo de origen.

```
public static SalaParaElPago obtenerPrimeraCitaDesdeArchivo(String rutaArchivo) {
    try (FileReader fr = new FileReader(fileName: rutaArchivo);
        BufferedReader br = new BufferedReader(in: fr)) {
        String linea;
        if ((linea = br.readLine()) != null) {
            String nombre = obtenerCampo(linea, indice: 0);
            String apellido = obtenerCampo(linea, indice: 1);
            String identificacion = obtenerCampo(linea, indice: 2);
            String ticket = obtenerCampo(linea, indice: 3);

            return new SalaParaElPago(nombreUsuario: nombre, apellidoUsuario: apellido, identificacionUsuario: identificacion, ticketCita: ticket);
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
    return null;
}
```

Clase SalaParaElPago. Esta clase tiene sus correspondientes getters and setters para el manejo de obtención y establecimiento de información con base a una cita en específico también contiene su debido constructor y toString para traducir la información que se maneja internamente a texto legible para el usuario y grabado de esta.

```
package com.asistentemedico.asistentedecitas.logica.PagoDeCitas;

public class SalaParaElPago {

    private String nombreUsuario;
    private String apellidoUsuario;
    private String identificacionUsuario;
    private String ticketCita;

    public SalaParaElPago(String nombreUsuario, String apellidoUsuario, String identificacionUsuario, String ticketCita) {
        this.nombreUsuario = nombreUsuario;
        this.apellidoUsuario = apellidoUsuario;
        this.identificacionUsuario = identificacionUsuario;
        this.ticketCita = ticketCita;
    }

    public String getNombreUsuario() {
        return nombreUsuario;
    }

    public void setNombreUsuario(String nombreUsuario) {
        this.nombreUsuario = nombreUsuario;
    }

    public String getApellidoUsuario() {
        return apellidoUsuario;
    }

    public void setApellidoUsuario(String apellidoUsuario) {
        this.apellidoUsuario = apellidoUsuario;
    }
}
```

Método marcarCitaComoPagada. Perteneciente a la clase GestorCitas este método se encarga de cargarCitasDesdeArchivo en la ruta donde se almacenan las citas del usuario en específico con base a su ticket manipulada la información por una lista doblemente enlazada y con los parámetros de la clase Cita que permite el manipulado de información en tiempo en curso en memoria donde algunos de estos parámetros son nombre, apellido, identificación, identificador, ticket, costo, doctor, estado, especialidad, asistencia, motivo. Las cargar los datos de la cita parametrizados en un nodo y enlazados uno al otro se busca el que queremos marcar como pago teniendo como insumo el ticket suministrado por el usuario. Y establecemos esa cita ubicada en un nodo como “PAGA” y reescribimos la información en la persistencia de datos.

```
public static void marcarCitaComoPagada(String ticket){

    DoblementeEnlazada<Cita> citas = cargarCitasDesdeArchivo(rutaArchivo: RUTA_ARCHIVO);
    Nodo<Cita> nodoActual = citas.getCabeza();

    while(nodoActual != null){
        Cita citaActual = nodoActual.getValor();
        if(citaActual.getTicket().equals(ticket)){
            citaActual.setEstado("Paga");
        }
        nodoActual = nodoActual.getSiguiente();
    }

    try(BufferedWriter writer = new BufferedWriter(new FileWriter(fileName: RUTA_ARCHIVO))){
        nodoActual = citas.getCabeza();

        while(nodoActual != null){
            Cita cita = nodoActual.getValor();
            writer.write(cita.toString() + "\n");
            nodoActual = nodoActual.getSiguiente();
        }
    }catch(IOException e){
        e.printStackTrace();
    }
}
```

Método de eliminarAutorizacionesDeSala. Este método se encarga de eliminar las citas que se buscan pagar y que ya han sido pagas dentro de la sala de espera. Despejando la saturación de usuarios. Con la implementación de nuestras propias estructuras podemos solamente eliminar del inicio el nodo ya que esta cola se atiende por orden de llegada y así mantenemos un correcto funcionamiento por parte del operador solamente atendiendo el pago de la primera cita y así consecutivamente.

```
public static void eliminarAutorizacionDeSala() {  
  
    try{  
        DoblementeEnlazada<String> autorizaciones =  
            readAutorizacionesFromFile();  
        autorizaciones.eliminarDelInicio();  
        writeAutorizacionesToFile2(authorizaciones);  
  
    } catch (IOException e) {  
    }  
}
```

Método iniciarTemporizador. Este método crea un temporizador que ejecuta una tarea periódicamente utilizando un “ScheduledExecutorService”. Y que es un “ScheduledExecutorService” bueno es también llamado “scheduler” utilizando “executors.new” esto crea un ejecutor de subprocesos programados con un solo hilo para ejecutar tareas periódicas.

Se llama al método del scheduler para programar la ejecución periódica de una tarea. Los parámetros de este método son. Una lambda definiendo la tarea a ejecutar. En este caso, la tarea consiste en llamar a los métodos saberEstado y moverPrimerPago que el método moverPrimerPago fue desglosado con anterioridad y pasarAlFramePagoExitoso en ese orden, pero solo si el estado del operador es flujo continuo es decir si el operador que gestiona las citas desea pasar al siguiente con forme ve la cola de pago. Con un retraso inicial de 1 unidad de tiempo que es traducido a un minuto antes de que comience la primera ejecución de la tarea y con un intervalo de casi 2 minutos entre cada ejecución automática.

```
}  
private void iniciarTemporizador() {  
    ScheduledExecutorService scheduler = Executors.newScheduledThreadPool(corePoolSize: 1);  
    scheduler.scheduleAtFixedRate(() -> {  
        saberEstado();  
        if(estadoOperador.equals(anObject: "Flujo Continuo")) {  
            moverPrimerPago();  
            pasarAlFramePagoExitoso();  
        }  
    }, initialDelay: 1, period: 2, unit: TimeUnit.MINUTES);  
}
```

Método de saberEstado. Encargado de obtener el estado del operador encargado para manejar el flujo del pago de citas médicas. Inicialmente se utiliza el método “estado” para obtener el estado de atención del operador en la sala de atención médica, utilizando la constante u ruta designada como parámetro. Este método realiza una operación de acceso a datos para recuperar el estado con la parametrización de datos con base a la clase EstadoParaPagoDeCitasMedicas. Se verifica si el estado de atención obtenido no es nulo, y obtenemos la información designándola en las variables que describen lo que contiene.

```
private void saberEstado() {  
    EstadoParaPagoDeCitasMedicas estadodeAtencion =  
    estado(rutaArchivo: RUTA_SALA_DE_ATENCION);  
    if(estadodeAtencion != null) {  
        nombreOperador = estadodeAtencion.getNombre();  
        apellidoOperador = estadodeAtencion.getApellido();  
        identificacionOperador = estadodeAtencion.getIdentificacion();  
        estadoOperador = estadodeAtencion.getEstadoDeFlujo();  
    }  
}
```


Método pasarAlFramedePagoExitoso. El método se encarga de verificar por usuario el identificador que contiene su cita si es válido es decir si ya ha sido pago, pasando a un Frame mostrando un nuevo marco de pago exitoso.

Su funcionamiento se inicializa una variable como false que indica el marco de pago exitoso ya ha sido mostrado. Inicialmente se establece. Se obtiene por medio del identificador obtenido en el recuadro de ticket se verifica si el marco de pago exitoso aún no ha sido mostrado, es decir, si FrameMostrado es false. Si el marco aún no ha sido mostrado se llama al método verificaridentificadordeautorizacion(identificador) del “GestorCitas” para verificar si el identificador de la cita de pago de citas es válido.

Si el identificador es válido (es decir, si “pasoalotroframe” es “true”) se crea una nueva instancia del marco como “CitaPagada” pasándole el identificador como parámetro. Luego, se cierra el marco actual con this.dispose y se hace visible el nuevo marco. También se configuran la ubicación y la capacidad de redimensionamiento del nuevo marco.

```
private boolean frameMostrado = false;
private void pasarAlFramedePagoExitoso() {

    String identificador = ticketEntrada.getText();

    if(!frameMostrado){
        boolean pasoalotroframe =
        GestorCitas.verificaridentificadordeautorizacion(ticket:identificador);

        if(pasoalotroframe==true){
            CitaPagada Exit = new CitaPagada(ticket:identificador);
            this.dispose();
            Exit.setVisible(b: true);
            Exit.setLocationRelativeTo(c: null);
            Exit.setResizable(resizable:false);
            frameMostrado=true;
        }
    }
}
```

Este método se encarga de cargar una lista doblemente enlazada de citas médicas desde la ruta especificada utilizando el método `cargarCitasDesdeArchivo` esta lista contiene información sobre todas las citas médicas almacenadas, pagas y no pagas en el sistema. Se inicializa un nodo como actual que apunta al primer nodo de la lista de citas. Se itera a través de la lista de citas médicas mientras el nodo actual no sea nulo. Dentro del bucle, se obtiene la cita médica actual desde el nodo actual. Se verifica si el ticket de la cita médica actual que le pasemos con base al usuario es igual al ticket pasado como parámetro y si el estado de la cita es "PAGA". Si ambos criterios cumplen, se devuelve true, lo que indica que el identificador de autorización es válido dando paso al otro Frame y la cita ha sido pagada. Si no se encuentra ninguna cita médica con el ticket dado que esté pagada, el bucle continúa iterando hasta el final de las citas. Si no la encuentra devuelve "false".

```
public static boolean verificaridentificadordeautorizacion(String ticket){  
  
    DoblementeEnlazada<Cita> citas = cargarCitasDesdeArchivo(rutaArchivo: RUTA_ARCHIVO);  
    Nodo<Cita> nodoActual = citas.getCabeza();  
  
    while(nodoActual != null){  
        Cita citaActual = nodoActual.getValor();  
        if(citaActual.getTicket().equals(anObject: ticket) &&  
            citaActual.getEstado().equalsIgnoreCase(anotherString: "Paga")){  
            return true;  
        }  
        nodoActual = nodoActual.getSiguiente();  
    }  
    return false;  
}
```

Registro de autorizaciones. AutorizarExamenActionPerformed encargado de manejar la acción de autorizar exámenes médicos en la interfaz de usuario. Inicialmente con la verificación de campos y selección de exámenes cabe resaltar que si es un examen que viene de una cita médica ya atendida por el usuario se selecciona automáticamente una cita de tipo examen se escoge el examen por voluntad si la cita que tuvo el usuario con anterioridad es si es una cita de tipo control. Inicialmente este código verifica los campos la validación del inicio de sesión y el proceso de autorización si la cita está pagada, el paciente ha asistido y no hay autorización previa o igual, se procede con la autorización de los exámenes seleccionados.

Se recopila información como el nombre del usuario, identificador de la cita, costo de los exámenes, descripción , etc. Se abre una nueva ventana para filtrar la autorización de los exámenes, mostrando esta información para su revisión y generando un ticket de autorización.

```
private void AutorizarExamenActionPerformed(java.awt.event.ActionEvent evt) {  
    verificarTicket();  
  
    boolean examenSeleccionado = examen1.isSelected() ||  
    examen2.isSelected() || examen3.isSelected() ||  
    examen4.isSelected() || examen5.isSelected() ||  
    examen6.isSelected();  
    if(camposCompletos() && examenSeleccionado)  
    {  
        String nombreUsuario= nombreEntrada.getText();  
        String apellidoUsuario = ApellidoEntrada.getText();  
        String identificacion = IdentificacionEntrada.getText();  
        String contrasenia = new String(value: ContraseniasEntrada.getPassword());  
        String identificadorCita = IdentificacionDeCitaEntrada.getText();  
        String ticket = TicketEntrada.getText();  
  
        ControladorUsuario.cargarUsuarios();  
        if(ControladorUsuario.validarInicioSesion(nombre:nombreUsuario,  
            apellido: apellidoUsuario, identificacion, contrasenia))  
        {  
            boolean citaPaga=GestorCitas.verificarCitaPagada  
            (identificadorCita, ticket);  
            boolean citaAsistida=GestorCitas.verificarAsistenciaCita  
            (identificadorCita, ticket);  
            boolean autorizada=GestorAutorizacion.verificarAutorizacion(  
            identificadorCita, ticket);
```

```

if(citaPaga && citaAsistida && !autorizada)
{
String especialidadMotivoSeleccionados = "";
if (examen1.isSelected()) especialidadMotivoSeleccionados
    = examen1.getText();
if (examen2.isSelected()) especialidadMotivoSeleccionados
    = examen2.getText();
if (examen3.isSelected()) especialidadMotivoSeleccionados
    = examen3.getText();
if (examen4.isSelected()) especialidadMotivoSeleccionados
    = examen4.getText();
if (examen5.isSelected()) especialidadMotivoSeleccionados
    = examen5.getText();
if (examen6.isSelected()) especialidadMotivoSeleccionados
    = examen6.getText();

int costo = generarCosto();

String descripcion = generarDescripcion();

int identificadordeautorizacion = generarIdentificadordeAuto();

String ticketAutorizacion = generarTicket();

this.dispose();
FiltroDeAutorizacion sala =
new FiltroDeAutorizacion(
nombreUsuario, apellidoUsuario, identificacion,
identificadorDeCita:identificadorCita,ticket,
especialidadMotivoSeleccionados, costo,descripcionPrincipal:descripcion,
identificadordeautorizacion, ticketAutorizacon:ticketAutorizacion
);

```

```

sala.setVisible(b: true);
sala.setLocationRelativeTo(c: null);
sala.setResizable(resizable: false);
}
else{
    String verificarExamen = "";
    if (examen1.isSelected()) verificarExamen =
        examen1.getText().trim();
    if (examen2.isSelected()) verificarExamen =
        examen2.getText().trim();
    if (examen3.isSelected()) verificarExamen =
        examen3.getText().trim();
    if (examen4.isSelected()) verificarExamen =
        examen4.getText().trim();
    if (examen5.isSelected()) verificarExamen =
        examen5.getText().trim();
    if (examen6.isSelected()) verificarExamen =
        examen6.getText().trim();

    boolean examenTipoCorrecto =
    GestorAutorizacion.verificarExamen(tipoExamen: verificarExamen);
    if(examenTipoCorrecto)
    {

        String costo = GestorAutorizacion.traerCosto(nombreUsuario,
        apellidoUsuario, identificacionUsuario: identificacion,
        identificadorDeCita:identificadorCita, ticket);

        String identificadorDeAutorizacion =
        GestorAutorizacion.traerIdentificadordeAutorizacion(
        nombreUsuario, apellidoUsuario,
        identificacionUsuario: identificacion, identificadorDeCita:identificadorCita, ticket);

```

```

        String costo = GestorAutorizacion.traerCosto(nombreUsuario,
        apellidoUsuario, identificacionUsuario: identificacion,
        identificadorDeCita:identificadorCita, ticket);

        String identificadorDeAutorizacion =
        GestorAutorizacion.traerIdentificadordeAutorizacion(
        nombreUsuario, apellidoUsuario,
        identificacionUsuario: identificacion, identificadorDeCita:identificadorCita, ticket);

        String traerDescripcion =
        GestorAutorizacion.traerDescripcion(nombreUsuario,
        apellidoUsuario, identificacionUsuario: identificacion,
        identificadorDeCita:identificadorCita, ticket);

        this.dispose();
        ustedYaCuentaConUnaAutorizacion check = new ustedYaCuentaConUnaAutorizacion
        (nombreUsuario, apellidoUsuario, identificacion,
        identificadorDeCita:identificadorCita,ticket,
        especialidadMotivosSeleccionados: verificarExamen, precio:costo,
        descripcion: traerDescripcion,identificadorDeAutorizacion);

        check.setVisible(b: true);
        check.setLocationRelativeTo(c: null);
        check.setResizable(resizable:false);
    }
    else{
        this.dispose();
        ErrorAlAgendar error = new ErrorAlAgendar();
        error.setVisible(b: true);
        error.setLocationRelativeTo(c: null);
        error.setResizable(resizable:false);
    }
    }
}

```

```
        else{
            this.dispose();
            ErrorAlAgendar error = new ErrorAlAgendar();
            error.setVisible(b: true);
            error.setLocationRelativeTo(c: null);
            error.setResizable(resizable:false);
        }
    }
    else{
        this.dispose();
        ErrorAlAgendar error = new ErrorAlAgendar();
        error.setVisible(b: true);
        error.setLocationRelativeTo(c: null);
        error.setResizable(resizable:false);
    }
}
```

Método “validarInicioSesion”. Este método se encarga de validar el inicio de sesión de un usuario en el sistema, comparando las credenciales proporcionadas con los usuarios almacenados. Se comienza buscando el ultimo usuario en la lista, recorriendo la lista de usuarios para encontrar una coincidencia exacta de nombre, apellido, identificación y contraseña. Resultado de la validación con un true si es correcto o false. Dependiendo para continuar con el flujo de la entrada a la sección de autorizaciones.

```
public static boolean validarInicioSesion(String nombre, String apellido, String identificacion, String contrasenia) {
    NodoUsuario actual = ultimo;
    if (actual == null) return false; // No hay usuarios cargados
    do {
        if (actual.getUsuario().getNombre().equalsIgnoreCase(anotherString: nombre)
            && actual.getUsuario().getApellido().equalsIgnoreCase(anotherString: apellido)
            && actual.getUsuario().getIdentificacion().equals(anObject: identificacion)
            && actual.getUsuario().getContrasenia().equals(anObject: contrasenia)) {
            return true; // Usuario encontrado, inicio de sesión exitoso
        }
        actual = actual.getSiguiente();
    } while (actual != ultimo);
    return false; // Usuario no encontrado, inicio de sesión fallido
}
```


Método “verificarCitaPagada”. En particular especificado para verificar si la cita está pagada en el sistema para proceder con el registro de la autorización del examen que desea hacerse el usuario. Iniciando por una carga de citas desde el archivo. Iterando por medio de las citas. Comenzando desde la cabeza de la lista. Verificando la cita se compara el identificador y el ticket de cada cita con los valores proporcionados como parámetros. Además, se verifica si el estado de la cita “Paga” el resultado de esta búsqueda devuelve true en caso de éxito y false si no.

```
public static boolean verificarCitaPagada(String identificadorCita, String ticket){

    DoblementeEnlazada<Cita> citas = cargarCitasDesdeArchivo(rutaArchivo: RUTA_ARCHIVO);
    Nodo<Cita> nodoActual = citas.getCabeza();

    while(nodoActual != null){

        Cita citaActual = nodoActual.getValor();

        if(citaActual.getIdentificador().equals(anObject: identificadorCita) &&
            citaActual.getTicket().equals(anObject: ticket) &&
            citaActual.getEstado().equalsIgnoreCase(anotherString: "Paga")){
            return true;
        }

        nodoActual = nodoActual.getSiguiente();
    }

    return false;
}
```

10. Especificación de Requerimientos

ID	REQUERIMIENTO	DESCRIPCIÓN	PRIORIDAD
110	Registro de Usuarios por Número Telefónico	Como usuario, quiero poder registrar mi información utilizando mi número telefónico como identificador. En caso de no contar con un número telefónico, deseo tener la opción de utilizar un identificador de llamadas para acceder al sistema.	MEDIA
220	Interfaz Dinámica para el Inicio de Sesión	Como usuario, quiero una interfaz de inicio de sesión dinámica y sencilla, para que las distintas características del programa están organizadas en páginas y se facilite la navegación.	MEDIA
330	Configuración de Tipo de Cita	Como usuario, quiero tener una casilla en la interfaz donde pueda configurar mi cita, para poder seleccionar el tipo de cita que deseo, ya sea general o de un área específica.	MEDIA
440	Selección de Tipo de Cita y Categoría	Como usuario, quiero poder seleccionar el tipo de cita que deseo y categoría, y tener la opción de decidir si pagaré la cita de una vez o después, para tener mayor flexibilidad y eficiencia en los tiempos de pago.	ALTA
550	Confirmación de Cita	Como usuario, deseo poder confirmar mi cita a través del sistema y que el sistema genere automáticamente el total a pagar, para agilizar el proceso de pago de las citas.	MEDIA
660	Modificación de Cita Designada	Como usuario, quiero poder modificar mi cita designada según el motivo y el tipo de cita que seleccione, para tener la comodidad de poder editar las especificaciones de la cita acorde al tipo de atención médica que deseo.	ALTA

770	Agendar Citas Médicas	Como usuario, deseo poder agendar citas médicas una vez iniciada la sesión en el sistema y tener la opción de seleccionar lo necesario para mi cita, así como la especialidad médica requerida. Además, deseo recibir una confirmación de la cita una vez que haya sido programada con éxito.	ALTA
880	Acceder a la Sala de Espera	Como usuario, deseo tener la capacidad de acceder a la sala de espera del centro de salud. Quiero poder verificar mi cita programada y, si es necesario, unirse a la sala de espera virtual para esperar mi turno de atención médica. Además, deseo recibir notificaciones o recordatorios cuando sea mi turno de ser atendido, para que haya una mejor organización en la sección de sala de espera.	ALTA
990	Autorización de Exámenes desde el Lobby	Como usuario, deseo tener la opción de acceder a la autorización de exámenes desde el lobby del centro de salud. Quiero poder revisar los exámenes recomendados por mi médico y autorizar su realización a través del sistema. Además, deseo recibir información sobre la preparación necesaria para cada examen y los detalles sobre cómo proceder después de su realización para estar mejor orientado y notificado acerca de las recomendaciones y cómo proceder con esta.	ALTA
101	Pago de citas	Como usuario, deseo tener la opción de realizar el pago de mi cita médica una vez establecida las especificaciones sobre el tipo de cita que quiero agendar, para cumplir de una forma óptima el respectivo pago antes de proceder con la cita médica o examen programado	ALTA
110	Atención de cita médica o examen	Como usuario, quiero tener la opción de elegir que doctor deseo que atienda mi cita médica o examen siempre y cuando esté libre y haga parte de los médicos autorizados a atender ese tipo de cita o examen, pero en caso de agendar un examen hacer tener antes la respectiva cita de control necesaria para la autorización del agendamiento de este, para tener una mayor flexibilidad y comodidad a la hora de ser atendido.	MEDIA
120	Cancelación de citas y exámenes	Como usuario, quiero tener la capacidad de cancelar mi cita médica en caso de que ocurra alguna situación por la que no pueda ser atendido, y que me devuelvan el 70% de lo pagado, para poder recibir una gran parte de lo pagado y tener la ventaja de usarlo para reagendar otra cita o examen.	ALTA

11. Implementaciones

En el proyecto Aurora, las implementaciones representan el puente crucial entre la concepción de ideas y la materialización de soluciones tangibles. Cada implementación no solo implica la aplicación de tecnologías y herramientas específicas, sino también una cuidadosa planificación y ejecución para garantizar que las funcionalidades diseñadas cumplan con los requisitos del usuario y las expectativas del proyecto.

Las implementaciones en este proyecto se abordaron con un enfoque centrado en la eficiencia y la calidad, priorizando la entrega oportuna de funcionalidades clave sin comprometer la estabilidad del proyecto ni la escalabilidad del sistema.

Se establecieron procesos de desarrollo ágiles que fomentaron la colaboración entre los equipos, la iteración continua y la adaptación rápida a los cambios en los requisitos del proyecto.

Continuando con la importancia de las correctas implementaciones y el correcto abordaje en este proyecto, las implementaciones son el motor que impulsa el progreso y la evolución del sistema de gestión de citas médicas. Cada una de las importantes características, mejora y corrige los errores llevados a cabo mediante las implementaciones efectivas contribuye a fortalecer la robustez y la utilidad del sistema.

11.1 Tecnologías

En el proyecto Aurora, se emplearon diversas tecnologías a destacar para dar vida al sistema de gestión de citas médicas, cada una desempeñando un papel crucial en la construcción del proyecto y operación del sistema.

Estas tecnologías abarcan de forma eficiente desde el lenguaje de programación Java hasta plataformas de desarrollo y herramientas de persistencia de la información.

- 1. Entorno de Desarrollo y Lenguaje de Programación:** Se utilizó Java como el lenguaje de programación principal debido a su robustez, portabilidad y amplia adopción en el desarrollo de aplicaciones empresariales. Además, se empleó NetBeans como entorno de desarrollo integrado (IDE), aprovechando su conjunto de herramientas avanzadas para agilizar el proceso de desarrollo y mejorar la productividad del equipo.
- 2. Interfaz de Usuario (UI),** Se creó un frontend para la aplicación de escritorio utilizando Java Swing, una biblioteca gráfica que ofrece una amplia gama de componentes visuales y una fácil integración con el entorno de desarrollo NetBeans. Esto permitió diseñar interfaces de usuario intuitivas y atractivas para mejorar la experiencia del usuario.
- 3. División de clases por Responsabilidad Única (SRP),** Se aplicó el principio de responsabilidad única para dividir las clases en unidades cohesivas y de responsabilidad única. Esto facilitó el mantenimiento del código, mejoró su legibilidad y promueve la reutilización de componentes, lo que contribuyó a la escalabilidad y mantenibilidad del sistema.

4. **Estructuras de Datos,** Se implementaron diversas estructuras de datos para el correcto funcionamiento y la dependencia de otras estructuras como ArrayList, HashMap, List, Arreglos, entre otros. Se implementó como doblemente Enlazada, pilas, colas y listas sencillas. Para organizar y manipular eficientemente la información relacionada con citas médicas, pacientes y médicos, exámenes, autorizaciones. Entre otros.
5. **Clases para manejo de Parámetros,** Se desarrollaron clases específicas para gestionar los parámetros de la información, lo que facilitó la manipulación y el acceso a los datos de manera estructurada y coherente. Esto contribuyó a la claridad y coherencia del código, así como a la escalabilidad del sistema.
6. **Persistencia de la información con archivos de Texto,** La persistencia de la información se llevó a cabo utilizando archivos de texto, que ofrecen una solución simple y efectiva para almacenar y recuperar datos sin la necesidad de un sistema de gestión de base de datos complejo. Escogimos algo práctico y un sistema de citas portable en cualquier sistema sin descargar otras características externas.
7. **Junit Test,** Se utilizaron pruebas unitarias con JUnit para garantizar la calidad y el correcto funcionamiento del código. Estas pruebas permitieron la detección temprana de errores y facilitaron la refactorización del código, asegurando la estabilidad y fiabilidad del sistema en todas las etapas del desarrollo.
8. **Control de versiones,** Se utilizó Git como sistema de control de versiones para gestionar el código fuente del proyecto. Git proporcionó un historial completo de cambios, permitiendo al equipo de desarrollo colaborar de manera efectiva y mantener un registro detallado de todas las modificaciones realizadas en el código. GitHub se empleó como plataforma de alojamiento remoto para el repositorio Git del proyecto. Esta elección permitió al equipo de desarrollo acceder al código desde cualquier lugar, facilitando la colaboración entre miembros del equipo ubicados en diferentes ubicaciones geográficas.

11.2 Software

El software implementado diferentes herramientas de software y estructuras utilizadas durante el desarrollo del proyecto.

El propósito del software. Principalmente es automatizar y optimizar el proceso de gestión de citas médicas en un entorno hospitalario o clínico. Facilitando la coordinación entre pacientes y médicos, garantizando una distribución eficiente de los recursos y una atención médica oportuna.

Funcionalidades Principales

- Gestión de citas médicas, Registro de pacientes, Asignación de médicos, gestión de exámenes, pago de citas, autorizaciones médicas. Entre otros.

Arquitectura

- El software sigue una arquitectura basada en capas, con una capa de presentación (UI) desarrollada en Java Swing para la interfaz de usuario de la aplicación de escritorio.
- La lógica está encapsulada en clases Java que implementan principios de diseño como la Responsabilidad Única (SRP) y la cohesión.
- Se utilizan estructuras de datos como listas doblemente enlazadas y clases específicas para el manejo de parámetros y estados de flujo.

Capa de presentación (UI)

- La interfaz de usuario se desarrolló con distintos componentes como Lbales, Tables, Botton, etc. El diseño de la interfaz se hizo de forma práctica y funcional para que cualquier tipo de usuario pudiera operar. También el manejo de eventos donde se desempeñaba un papel crucial si el usuario digita una contraseña errónea no simplemente quedarse en esa pestaña por el contrario tener un evento pasando a una sección de contraseñas incorrectas y su debida vinculación a acciones correspondientes. Finalmente, la accesibilidad y usabilidad según su acción o botón describe lo que se realiza en esa parte del programa.
- Se han implementado controladores de eventos que responden a las acciones del usuario, como clics de botón o selección de elementos, para ejecutar las acciones correspondientes, como programar citas o realizar pagos. Se ha priorizado la respuesta rápida y la retroalimentación visual para mejorar la interactividad.

Capa de lógica

- La capa de lógica del software incluye clases como FlujoDeSalas, que se encargan de gestionar el flujo de operaciones en las diferentes salas del sistema, como la sala de atención de citas, la sala de pagos y la sala de autorizaciones. Estas clases implementan métodos para cargar, modificar y guardar el estado de las salas, así como para realizar acciones específicas en función de los eventos del sistema.
- Las clases de la capa de lógica siguen los principios de diseño como la Responsabilidad Única (SRP) y la cohesión, donde cada clase tiene una única responsabilidad y está altamente relacionada con su función específica en el sistema. Se han evitado acoplamientos innecesarios entre clases para mejorar la mantenibilidad y la reutilización del código.
- La lógica del software se ha diseñado con una clara separación de preocupaciones y una estructura modular, donde cada componente se encarga de una tarea específica y puede ser fácilmente reemplazable o extendido sin afectar otras partes del sistema. Esto permite una mayor flexibilidad y escalabilidad en el desarrollo y mantenimiento del software.

Estructuras de datos y clases específicas

- Listas doblemente enlazadas jugó un papel crucial en este proyecto al igual que las pilas, colas y listas sencillas. En la implementación del software, se utiliza una lista doblemente enlazada para mantener el flujo de estados en las diferentes salas del sistema. Esta estructura de datos proporciona un acceso eficiente tanto hacia adelante como hacia atrás en la lista, lo que es crucial para manejar el flujo de operaciones en las salas, como cambiar el estado de una cita médica o un proceso de autorización.
- El software incluye clases específicas como EstadoParaSalaDeCitasAtencionMedica, Entre otras. Para representar y manejar los diferentes estados de flujo en las salas del sistema. Todo esto siendo posible gracias a las listas dobles donde estas clases encapsulan la información relevante, como nombres, identificación, estado y proporciona métodos para la debida manipulación de la información coherente y segura.

Patrones de diseño

- Se tuvieron en cuenta los patrones de diseño como el patrón de diseño Modelo-Vista-Controlador (MVC). Ya que este patrón separa la lógica (modelo) de la presentación(Vista) y la lógica de control (Controlador), lo que facilita la modularidad y la reutilización del código.
- La aplicación de patrones de diseño para este proyecto funcionó para estructurar el código ayudando a organizar y estructurar el código de manera efectiva, separando las preocupaciones y facilitando la comprensión y mantenimiento del software. Por ejemplo, utilizando el patrón MVC, se puede dividir el código en componentes independientes como se desarrolló que son más fáciles de mantener y extender.

Escalabilidad, flexibilidad, Desempeño y optimización

- La arquitectura del software está diseñada para facilitar la escalabilidad, permitiendo la incorporación de nuevas funcionalidades y la expansión del sistema para adaptarse a cambios en los requisitos del negocio o del usuario. Esto se logra mediante una estructura modular y una clara separación de preocupaciones que permite agregar nuevas capas o componentes sin afectar el funcionamiento existente del sistema.
- Se han tenido en cuenta las consideraciones de flexibilidad en el diseño del software, asegurando que el sistema pueda adaptarse fácilmente a cambios en los requisitos o tecnologías.
- En la arquitectura del software, se han implementado estrategias de optimización para mejorar el rendimiento del sistema. Esto puede incluir técnicas como la optimización de consultas, la minimización de la carga en la interfaz de usuario y la optimización de algoritmos para reducir el tiempo de procesamiento.

11.3 Pruebas

Se tuvieron en cuenta distintos estados de prueba como pruebas de integración, escenarios de uso, Junit test para los métodos, Resultados de prueba y validaciones.

- Pruebas de unidad

Se utilizaron validaciones de comportamiento individual de componentes o unidades de código en el módulo de gestión de usuarios. Una prueba de estas fue la verificación que no permitiera grabar dos usuarios con la misma identificación que se centraba en probar cada clase que diera al cumplimiento de este requerimiento y método de forma aislada para asegurar su correcto funcionamiento.

- Framework de pruebas utilizado

Se empleó Junit para escribir y ejecutar las pruebas de unidad en el modelo de gestión de usuarios.

Algunos ejemplos de estas pruebas fueron, cargar el flujo de sala de atención, cambiar el estado de sala de atención, detener el flujo de atención de citas.

```
@Test
public void testCargarFlujoDeSalaDeAtencion() {
    DoblementeEnlazada<EstadoParaSalaDeCitasAtencionMedica> flujo;

    flujo = FlujoDeLasSalas.cargarFlujoDeSalaDeAtencion(rutaArchivo: RUTA_PRUEBA_ATENCION);

    assertNotNull(actual: flujo);
    assertFalse(condition: flujo.estaVacia());
}

@Test
public void testCambiarEstadoDeSalaDeAtencionMedicaAFlujoContinuo() {
    String nombre = "Juan";
    String apellido = "Pérez";
    String identificacion = "12345";

    FlujoDeLasSalas.cambiarEstadoDeSalaDeAtencionMedicaAFlujoContinuo(nombre, apellido, identificacion);
}
```

```

@Test
public void testCambiarEstadoDeSalaDeAtencionMedicaAFlujoDetenido() {
    String nombre = "Juan";
    String apellido = "Pérez";
    String identificacion = "12345";

    FlujoDeLasSalas.cambiarEstadoDeSalaDeAtencionMedicaAFlujoDetenido(nombre, apellido, identificacion);
}

@Test
public void testUsuarioExiste_CuandoUsuarioExiste_DeberiaRetornarTrue() {
    Usuario usuarioExistente = new Usuario(nombre:"Nombre", apellido: "Apellido", identificacion: "12345", edad: "", telefono: "", contrasenia: "");

    boolean resultado = ValidadorUsuarioExistente.usuarioExiste(usuario: usuarioExistente);

    assertTrue(condition:resultado);
}

@Test
public void testUsuarioExiste_CuandoUsuarioNoExiste_DeberiaRetornarFalse() {
    Usuario usuarioNoExistente = new Usuario(nombre:"NombreNoExistente", apellido: "ApellidoNoExistente", identificacion: "IDNoExistente", edad: "", telefono: "", contrasenia: "");

    boolean resultado = ValidadorUsuarioExistente.usuarioExiste(usuario: usuarioNoExistente);

    assertFalse(condition:resultado);
}
}

```

```

@Test
public void testGuardarUsuario_DeberiaGuardarUsuarioCorrectamente() {
    Usuario usuario = new Usuario(nombre:"Nombre", apellido: "Apellido", identificacion: "ID", edad: "Edad", telefono: "Telefono", contrasenia: "Contrasenia");

    boolean resultado = ManejadorArchivo.guardarUsuario(usuario);

    assertTrue(condition:resultado);
    assertTrue(condition:usuarioExisteEnArchivo(usuario));
}

@Test
public void testGuardarUsuario_DeberiaRetornarFalseSiHuboError() {
    Usuario usuario = null; // Usuario inválido

    boolean resultado = ManejadorArchivo.guardarUsuario(usuario);

    assertFalse(condition:resultado);
}

```

- **Resultados esperados y obtenidos**

Se esperaba que todas las pruebas de unidad pasaran correctamente, validando así el comportamiento de las clases y métodos de distintos eventos de gestión de usuarios. Los resultados obtenidos mostraron que todas las pruebas pasaron sin errores, confirmando que el módulo funcionaba según lo esperado.

- **Pruebas de integración**

En este punto una vez se probaron los métodos por medio de Junit test para validar su funcionamiento interno se probó la interacción entre el módulo de gestión de usuarios y otros módulos del sistema. Testeando el programa y usándolo como sería un día a día normal. Donde se centró en probar la comunicación entre el módulo de usuarios y los módulos relacionados, como el de autenticación y el de permisos.

- **Escenario de prueba de integración**

Esta parte de prueba de integración fue la creación de usuarios y su autenticación en el sistema por medio de contraseña, verificando que los datos se sincronicen correctamente entre los diferentes módulos.

- **Resultados de pruebas de integración**

Los resultados finales y obtenidos de las pruebas de integración confirmaron que el módulo de gestión de usuarios funcionaba correctamente en conjunto con otros módulos del sistema. Se verificó correcta comunicación y sincronización de datos entre los diferentes componentes.

- **Pruebas de aceptación**

1. Las pruebas de aceptación se utilizaron para validar que el módulo de gestión de usuarios cumpliera con los requisitos y expectativas del cliente o usuario final. Se centraron en probar los casos de uso principales del sistema relacionados con la gestión de usuarios.
2. Los escenarios de pruebas de aceptación se definieron varios escenarios de prueba de aceptación que representaban las acciones principales que realizarían los usuarios en el sistema, como registrar un nuevo usuario, iniciar sesión y gestionar las citas y su correspondiente visualización de vida clínica.
3. Los resultados de las pruebas de aceptación confirmaron que el módulo de gestión de usuarios cumplía con los criterios de aceptación definidos por el cliente. Se verificó el funcionamiento del módulo y operó correctamente según lo esperado en distintos escenarios de uso y que satisficiera las necesidades y expectativas de los usuarios finales.

12. Cronograma

[illegible][illegible]

13. Resultados

Algunos de los resultados más destacados en este proyecto y con mayor impacto son los siguientes.

1. **Sistema de operación,** La implementación de un operador dedicado a coordinar el flujo de pacientes ha sido fundamental para mejorar la eficiencia operativa de nuestro sistema. Esta medida ha llevado a una reducción significativa en los tiempos de espera para la atención de diversos servicios, como pagos de citas, autorizaciones, exámenes y consultas médicas especializadas. La introducción de este enfoque ha permitido agilizar todo el proceso, facilitando una transición suave y eficiente entre las diferentes áreas de atención. Como resultado, los pacientes experimentan tiempos de espera más cortos y una atención más rápida y efectiva.
2. **Experiencia y funcionalidad integradas,** Para mejorar la experiencia del usuario, realizamos una actualización completa en la visualización y diseño de nuestro programa. Esta actualización se centró en adaptar la plataforma para ofrecer una experiencia fluida y altamente compatible con las demandas del mercado actual. Además de mejorar la satisfacción del usuario. La plataforma por su interfaz intuitiva y fácil de usar es práctica, lo que genera opiniones positivas.
3. **Cancelación efectiva y política de reembolso,** Hemos implementado un proceso eficiente para gestionar las cancelaciones de citas. Nuestro sistema permite a los usuarios cancelar citas de manera fácil y rápida, sin necesidad de interactuar con un operador. Además, hemos establecido una política de reembolso que devuelve el 80% del costo de la cita en caso de cancelación. Esta política no solo garantiza la satisfacción del cliente, sino que también promueve la responsabilidad en el uso de los servicios. Las citas no pagadas se eliminan automáticamente del sistema, y se proporciona un pantallazo de confirmación de cancelación. Garantizando una experiencia transparente y sin complicaciones.
4. **Integración de datos en la historia clínica y personalización de la atención,** Nuestro programa se integra de manera efectiva con las historias clínicas de los pacientes, lo que proporciona a los profesionales médicos un acceso rápido y fácil a la información relevante durante las consultas. Esta integración permite una atención más personalizada y eficiente, ya que se puede acceder a un panorama completo de la salud, movimientos, citas , etc. Además, hemos mejorado la capacidad de personalización de la atención al utilizar los datos para relacionarlos con un examen dejado después de la cita. Simplificando procesos lo que contribuye a una mejor calidad de atención y resultados para los usuarios.

- 5. Aplicación de estructuras de datos para optimizar procesos y persistencia de la información,** La eficiencia operativa de nuestro sistema se ha beneficiado significativamente de la aplicación de estructuras de datos adecuadas para optimizar los procesos y garantizar la persistencia de la información. La introducción de estructuras de datos como colas, ha permitido una gestión eficiente del flujo de pacientes y de los datos asociados a cada cita médica. Por ejemplo, hemos utilizado colas para gestionar el orden de atención de los pacientes en función de su llegada, garantizando una distribución equitativa de los recursos y una reducción en los tiempos de espera. Además, hemos implementado la persistencia de la información en archivos de texto para agilizar la búsqueda de información del sistema, lo que contribuye a una transición suave y eficiente entre las diferentes áreas de atención. En cuanto a la persistencia de la información, hemos utilizado estructuras de datos como lista doblemente enlazada, pila, cola, lista sencilla.
- 6. Mejora en la calidad de atención al usuario,** Todos los resultados mencionados anteriormente han contribuido a mejorar la calidad de atención que ofrecemos a nuestros usuarios. La optimización de la eficiencia operativa ha permitido reducir los tiempos de espera, lo que se traduce en una atención más rápida y efectiva. La experiencia y funcionalidad integradas han mejorado la satisfacción del usuario, lo que se refleja en las opiniones positivas de nuestros clientes. Además, la política de cancelación efectiva y la política de reembolso han demostrado nuestro compromiso con la satisfacción del cliente y la responsabilidad en el uso de nuestros servicios. La integración de datos en la historia clínica y la personalización de la atención han facilitado una atención más personalizada y eficiente, lo que contribuye a una mejor calidad de atención y resultados para nuestros usuarios.
- 7. Impacto en la eficiencia y productividad del personal médico,** Además de beneficiar a los usuarios, los resultados obtenidos también han impactado positivamente en la eficiencia y productividad del personal médico. La implementación de estructuras de datos eficientes ha optimizado los procesos internos del sistema, lo que ha permitido a los profesionales médicos acceder rápida y fácilmente a la información relevante durante las consultas. Esto se traduce en una atención más eficiente y en la capacidad de ofrecer un cuidado más personalizado a cada usuario. Además, la persistencia de la información en archivos de texto ha facilitado la búsqueda y recuperación de datos, lo que ha contribuido a una gestión más eficiente de la información médica.
- 8. Seguimiento y mejora continua,** Es importante destacar que estos resultados son el resultado de un esfuerzo continuo por mejorar y optimizar nuestro sistema. Seguiremos monitoreando y evaluando nuestros procesos para identificar áreas de mejora y garantizar que sigamos ofreciendo la mejor atención posible a nuestros usuarios. La aplicación de estructuras de datos eficientes y la persistencia de la información son solo el comienzo de nuestro compromiso con la innovación y la excelencia en el cuidado de la salud.

Además, se destacan los siguientes resultados complementarios

- Seguimiento
- Modularidad
- Incremento en la satisfacción
- Reducción de tiempos de espera
- Mayor accesibilidad a los servicios médicos
- Optimización de recursos y costos
- Aumento en la precisión diagnóstica
- Mayor integración y persistencia segura de datos
- Facilitación del seguimiento o historia clínica del usuario
- Confidencialidad de la información
- Mejora en la comunicación entre pacientes y profesionales de la salud
- Adaptabilidad a las necesidades cambiantes del usuario
- Cumplimiento de estándares y regulaciones del sector salud

El link del funcionamiento del programa es el siguiente:

<https://youtu.be/xT8jgIb5sGY?si=TvEozMcL7QnHkhAl>

y https://youtu.be/Ry6H0Bn9imQ?si=T2qua30_FqE6-weW

14. Conclusiones

Las conclusiones de este proyecto que tienen un alto impacto operativo, son los siguientes

1. **Impacto en la eficiencia operativa,** La implementación de medidas específicas, como la asignación de un operador dedicado y la optimización de los procesos, ha resultado en una notable mejora en la eficiencia operativa del sistema, reduciendo significativamente los tiempos de espera y agilizando los servicios prestados. Gracias a la ventaja de las estructuras de datos implementadas de una forma práctica y eficiente como las listas doblemente enlazadas.
2. **Mejora en la experiencia del usuario,** Las actualizaciones en el diseño y funcionalidad de la plataforma han contribuido a una experiencia del usuario más satisfactoria, reflejada en opiniones positivas y una mayor accesibilidad a los servicios médicos. Se comprendió distintos puntos de diseño para que vaya de la mano el diseño con lo práctico del programa.
3. **Políticas efectivas de cancelación y reembolso,** La implementación de políticas claras y eficientes para la cancelación de citas y el reembolso ha promovido la satisfacción del cliente y la responsabilidad en el uso de los servicios, garantizando una experiencia transparente y sin complicaciones. Yendo de la mano con el marco Legal haciendo una notable conclusión que una aplicación sin un marco legal estructurado y que cumpla con los estándares para funcionar correctamente no es un aplicativo funcional.
4. **Integración de datos y personalización de la atención,** La integración efectiva de datos en las historias clínicas de los usuarios ha permitido una atención más personalizada y eficiente facilitando el acceso a diferente información relevante durante consultas médicas. Como la visualización de la información de citas cuando el usuario solicita verla.
5. **Impacto positivo,** La elección y aplicación adecuada de las estructuras de datos, como las listas doblemente enlazadas, pilas, colas, etc. ha permitido un impacto positivo y una escalabilidad eficiente del sistema. Esto se traduce en la capacidad de manejar un crecimiento en la cantidad de datos y usuarios sin comprometer el rendimiento, además, el uso de métodos con responsabilidad única a pesar de estar relacionados en su lógica y el uso de estructuras de datos optimizadas ha simplificado las tareas de mantenimiento y desarrollo, facilitando la incorporación de nuevas funcionalidades y la corrección de errores de manera ágil y efectiva.
6. **Compromiso con la mejora continua,** Estos resultados y conclusiones son con base que con el suficiente esfuerzo continuo por mejorar y optimizar el sistema. Se seguirá monitoreando y evaluando los procesos para identificar áreas de mejora y garantizar que se continúe ofreciendo la mejor atención posible a los usuarios.

15. Referencias Bibliográficas

(Home) About Version Control-About Version Control. Software Freedom conservancy. <https://git-scm.com/book/es/v2/Inicio---Sobre-el-Control-de-Versiones-Acerca-del-Control-de-Versiones>

Wikipedia contributors. (s/f). Cola de prioridades. Wikipedia, The Free Encyclopedia. https://es.wikipedia.org/w/index.php?title=Cola_de_prioridades&oldid=145477000

Wikipedia contributors. (s/f). Modelo–vista–controlador. Wikipedia, The Free Encyclopedia. <https://es.wikipedia.org/w/index.php?title=Modelo%E2%80%93vista%E2%80%93controlador&oldid=138615253>

Wikipedia contributors. (s/f). Control de Versiones. Wikipedia, The Free Encyclopedia. https://es.wikipedia.org/w/index.php?title=Control_de_versiones&oldid=152003367

¿Qué es Git? (s/f). Microsoft.com. Recuperado el 3 de septiembre de 2023, de <https://learn.microsoft.com/es-es/devops/develop/git/what-is-git>

Tablado, F. (2020, septiembre 7). Bases de datos XML. Características y tipos. Ayuda Ley Protección Datos; AyudaLeyProteccionDatos. <https://ayudaleyprotecciondatos.es/bases-de-datos/xml/>

Wikipedia contributors. (s/f). Java Remote Method Invocation. Wikipedia, The Free Encyclopedia. https://es.wikipedia.org/w/index.php?title=Java_Remote_Method_Invocation&oldid=148805182

Base de Datos TXT : Parte I - Perl En Español. (s/f). Perlenespanol.com. Recuperado el 3 de septiembre de 2023, de https://perlenespanol.com/tutoriales/bases_de_datos/base_de_datos_txt_parte_i.html

Software development kits and command line interface. (s/f). Oracle.com. Recuperado el 9 de mayo de 2023, de <https://docs.oracle.com/en-us/iaas/Content/API/Concepts/sdks.htm>

UML Class Diagram tutorial. (s/f). Visual-paradigm.com. Recuperado el 3 de septiembre de 2023, de <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/>

NetBeans, A. (s/f). Designing a Swing GUI in NetBeans IDE. Apache.org. Recuperado el 3 de septiembre de 2023, de <https://netbeans.apache.org/kb/docs/java/quickstart-gui.html>

What is GitHub? A beginner's introduction to GitHub. (2018, abril 20). Kinsta®; Kinsta. <https://kinsta.com/knowledgebase/what-is-github/>

Deloitte (Sin fecha). *Qué es el desarrollo en espiral*. Deloitte.com. Recuperado el 17 de febrero de 2024, de <https://www2.deloitte.com/es/es/pages/technology/articles/que-es-el-desarrollo-en-espiral.html>

Ar-racking.com (2019, julio 26). *Método LIFO de Gestión Stock: Qué es y cuando se utiliza*. <https://www.ar-racking.com/co/blog/metodo-lifo-de-gestion-stock-que-es-y-cuando-se-utiliza/>

Arnold, K., Gosling, J., & Holmes, D. (Sin fecha). *THE java™ programming language, fourth edition*. Ase.ro. Recuperado el 29 de febrero de 2024, de <https://www.acs.ase.ro/Media/Default/documents/java/ClaudiuVinte/books/ArnoldGoslingHolmes06.pdf>

SYDLE. (s. f.). Post title | SYDLE. <https://www.sydle.com/blog/billing-rules-6400e02757aff34f9e0a3155>

Juviler, J. (2024, enero 5). *What is GitHub? (and for what is it used?)*. HubSpot. <https://blog.hubspot.com/website/what-is-github-used-for>

Programiz.com (Sin fecha). *Circular linked list*. Recuperado el 7 de marzo de 2024, de <https://www.programiz.com/dsa/circular-linked-list>

SYDLE. (s. f.). Post title | SYDLE. <https://www.sydle.com/blog/billing-rules-6400e02757aff34f9e0a3155>