Universidad de las Ciencias Informáticas

Facultad 3

Componentes para la gestión de alertas, mensajes y notificaciones en el Sistema de Gestión para la Atención a la Población

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autores:

Miguel Alejandro Diaz Rivera

Rafael Alejandro Aguilera Rodríguez

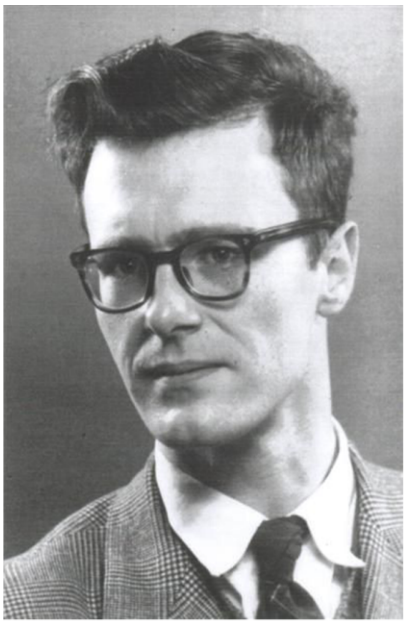
Tutores:

MSc. Yordanis García Leiva

Ing. Robin Sencial Terrero

Septiembre, 2020

“Año 62 de la Revolución”



"Raise your quality standards as high as you can live with, avoid wasting your time on routine problems, and always try to work as closely as possible at the boundary of your abilities. Do this, because it is the only way of discovering how that boundary should be moved forward."

Edsger Dijkstra.

DECLARACIÓN DE AUTORÍA

Declaramos que somos los únicos autores de este trabajo y autorizamos a la Facultad 3 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmamos la presente a los _____ días del mes de _____ del año _____.

_____

Miguel Alejandro Diaz RiveraRafael Alejandro Aguilera

_____ _____

MSc. Yordanis García Leiva Ing. Robin Sencial Terrero

AGRADECIMIENTOS

De Miguel:

Somos los primeros graduados en hacerlo bajo circunstancias como las que vivimos. De esto, cubanos al fin, muchos hacemos bromas frecuentemente. Somos la primera generación que no tendrá, al menos por ahora, un acto de graduación, un título, y donde todo lo que suceda de aquí en adelante será, por lo pronto, incierto. Somos una generación especial, pero esto no es un privilegio. No somos especiales en la manera en que quisiéramos. No somos una generación especial por nuestros logros como graduados, no somos una generación especial por nuestros logros como ingenieros, no somos una generación especial en la manera en que nos causaría orgullo, no somos una generación especial en la manera en que me gustaría a mícontarles a mis hijos. Somos, simplemente, una generación de graduados, que culmina sus estudios en una situación de innegable incertidumbre, y que para poder hacerlo necesitó, sin duda de ningún tipo, de varias personas, y como consecuencia, somos una generación, sobre todas las cosas, agradecidas, con eso, con varias personas.

Quiero comenzar con un gesto que considero sensible y necesario. Quiero agradecer a un grupo inmenso de personas con los cuales todos estamos, o deberíamos estar agradecidos. Que no participaron en nuestros trabajos de cursos, pero cuya labor, en mi opinión, y con el respeto de los demás graduados, trasciende de alguna manera cualquiera fueran nuestros esfuerzos para lograr lo que con alegría y orgullo todos celebramos hoy, nuestra graduación. Esas

personas son el personal de la salud de Cuba, que combate la covid-19 en estos mismos momentos en cada rincón de nuestro país y del mundo. Gracias.

Regresando a nuestro contexto en particular, hay varias personas con las cuales me siento agradecido y de las cuales me llevo para siempre un pedacito de ellos. Primero quiero ante todo agradecer a mi madre, muchas cosas sucedieron este año, no todas buenas, y no creo haber imaginado jamás cuánto se puede necesitar de alguien hasta que necesité de ella. Gracias mamá. Gracias a mi padre también por existir, por su preocupación constante a pesar de la distancia. Gracias papá.

En cuanto la tesis, hay una lista, que no es larga, pero es importante. Primero que todo quiero agradecer a dos personas, a mis tutores. Estas personas, que han sido de un apoyo incondicional a lo largo de este camino. Pero quiero comenzar con una persona en especial, y especial en sí misma, como le decimos nosotros "de oro", gracias profe Yordanis. Gracias por ser más que nuestro tutor nuestro amigo. Gracias por la guía, pero sobre todo gracias por la incondicionalidad. En esto puedo hablar sin temor, en nombre de mi compañero de tesis Rafael y yo, acerca de las incontables horas, de las horas tardes, de los incontables consejos, de los incontables "cocotazos" como dice el, que decía nos iba a dar cuando no realizábamos una tarea con la calidad que se requería. Gracias por todo, de veras, porque de esta más que un tutor, nos llevamos un amigo sincero como dijera Martí, sabemos ambos el privilegio que fue tener tutores como ustedes. De ahí el otro agradecimiento, gracias al profe Robin, por su apoyo también incondicional, por sus tormentas de idea con nosotros en el laboratorio, ideas que sentaron las bases de lo que sería nuestro proyecto.

Gracias por sus revisiones, y correcciones, siempre acertadas, a pesar de las barreras comunicativas que impone la actual situación sanitaria en el país. Quiero agradecer también a una persona inesperada aquí, a quien yo llamo mi tercera tutora, aunque ella se muera de la risa cuando lo hago, pues fueron incontables las preguntas e incondicionales las respuestas de Yamila Ortuño, la analista de nuestro centro, durante este periodo difícil, y sin las cuales hubiera sido casi imposible conceptualizar la aplicación de la manera en que se hizo. Gracias Yamila.

Gracias al tribunal, por estar aquí, por evaluarnos, gracias a todos. Gracias a los oponentes durante todo el proceso de elaboración de la tesis, gracias al profe Sandy, por sus revisiones y comentarios, gracias al profe Yoslenys, gracias al profe Fabra, al profe Reinier. Si menciono a algunos en particular es por mera cuestión de afecto quizás, pero gracias a cada uno de los oponentes a lo largo del trayecto, de cuyos señalamientos, correcciones y sugerencias es resultado el presente trabajo, sus observaciones estuvieron presentes en todo momento.

Gracias también, y, por último, a esos amigos, dentro y fuera de la universidad gracias a los cuales, la cuarentena no fue solo tesis y colas para el aceite. Esos que me escribieron para como dicen "saber si estaba vivo" o "saber si había cogido covid", entre otras variantes como el ampliamente conocido "que metes papa". No los voy a mencionar, tengo mucho miedo que se me quede alguno, pero ellos saben quiénes son, gracias.

De nuevo, gracias a todos.

De Rafael:

Siempre supe que los agradecimientos de mi tesis sería lo más difícil para mí, lo que nunca imaginé que sería en momentos tan difíciles para mi país, para el mundo y que la vida me impusiera un reto tan grande de perder a dos de mis seres queridos en menos de 5 meses a mi padre y mi abuelita, que hubieran estado muy orgullosos.

Quiero empezar agradeciendo a mi tribunal y mi oponente por guiarme en el transcurso de los seminarios de tesis, a todos gracias por brindarme su experiencia como profesionales. Además, agradecer a todos mis profesores y compañeros de la carrera, en especial a mis tutores Yordanys y Robin y a mi compañero de tesis Miguel Alejandro que sin el apoyo y dedicación de ellos no se hubiese logrado.

Quiero agradecer desde lo más profundo de mi corazón a todos los que han estado siempre apoyándome a mi madre, a mi segundo padre el Nana y a mi otra abuelita Ada, a mi madrina Bárbara, mis tías Yoanka y Fina, a mi tío Alexander y mis primos Alejandro y Carmen Laura, a mi padrino Carlos, a mi segunda madre Mariela, muchas gracias por todo. A mi familia por parte de padre.

También quería agradecer a mis suegros Eliecer y Elaidis, gracias por confiar en mí y abrirme las puertas de su casa y su corazón, y a todo el familión de mi novia gracias por todo el apoyo.

A mis amigos EL BERRACO (Grabiel), gracias mi hermano de corazón, al Andilucho, Gonza, Albert, Carlos Rafael, Jesús, El gurri, Leticia, Yelina yWendy, gracias por su amistad y su apoyo incondicional, a mis compañeros de FIFA y de momentos inolvidables Rolando, Ramses, el tanque (René), el kuko (Ricardo), Osorio, Eddy, les quiero un montón y muy especial Arian por su apoyo con la tesis en estos momentos de pandemia.

Llegó el momento de darle las gracias a una personita muy especial para mí, mi futura esposa y madre de mis hijos MELISSA, a ti que más que novia has sido mi amiga gracias por confiar en mí y darme tu amor, te amo mucho mi AMORZÓN.

Gracias Fidel y a esta Revolución que hicieron posible que en este año 2020, se cumpla mi sueño de ser Ingeniero en Ciencias Informáticas.

DEDICATORIA

De Miguel:

Quiero dedicarles estas palabras a mis abuelos ambos, Antonia Santos y Sabino Rivera, padres de mi madre, y personas que muy ancianos, y sin saberlo, me han enseñado tanto.

De Rafael:

Dedico mi tesis a mi papá Pedro Rafael, a mi abuela Gladys y mi mamá Odalys.

A ti papá, gracias por darme la vida, quiero que sepas que, a pesar de no estar entre nosotros, no hay un día de este mundo que deje de pensar en ti y en mi abuelita, me hubiese encantado que estuvieran presentes en este día tan especial, porque sé que el mayor deseo de ustedes era que me graduara, a ustedes donde quiera que estén, descansen en paz, ya cumplí, ya soy Ingeniero en Ciencias Informáticas, los amo mucho.

A ti mamá, gracias por darme la vida, gracias por educarme, gracias por confiar en mí y apoyarme incondicionalmente siempre, gracias por darme fuerzas en todo momento, gracias por preocuparte y tener fe en mí, gracias por ser la mejor madre del mundo, te amo mucho.

RESUMEN

El Sistema de Gestión para la Atención a la Población constituye una aplicación web desarrollada en el Centro de Gobierno Electrónico de la Universidad de las Ciencias Informáticasque tiene como objetivo crear un expediente único para cada persona que promueve un escrito de queja en un ministerio o institución pública. Una vez que el escrito es registrado, este pasa por un grupo de estados hasta que sea cerrado. Cada uno de los usuarios del sistema requiere ser notificado sobre los escritos que le sean asignados para ser procesados.En la actualidad, el sistema no brinda la posibilidad de notificarle a sus usuarios sobre la asignación de escritos. Tampoco alerta sobre los escritos que se encuentran pasados de término.Además, no existe una vía de comunicación entre los usuarios a través del propio sistema. La presente investigación tiene como objetivo desarrollar tres componentes para la gestión de alertas, mensajes y notificaciones en este software, que facilite el trabajo y la comunicación de los usuarios. El desarrollo de la solución está guiado por el uso de la metodología de desarrollo de software Proceso Unificado Ágil en su variación para la Universidad de Ciencias Informáticas y la utilización de tecnologías de código abierto. El resultado de la investigación fue validado aplicando pruebas de software. Además, se verifica el cumplimiento del objetivo general de la investigación a partir de la definición de un conjunto de indicadores que permiten evaluar la relación causa-efecto de la variable independiente sobre las variables dependientes de la investigación.

PALABRAS CLAVES: alerta, escrito, mensaje, notificación, usuario

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

Los ministerios e instituciones de la administración pública en Cuba, tienen entre sus funciones brindar atención a los planteamientos que realiza la población a través de escritos clasificados como quejas, denuncias, sugerencias o de otro tipo. En la actualidad, la vía mediante la cual los planteamientos llegan es diversa, así como las causas y forma de tramitación de estos. Todo lo anterior dificulta el proceso de atención a la población en cada una de las instituciones.

En el año 2018 el Centro de Gobierno Electrónico (CEGEL) de la Facultad 3 de la Universidad de las Ciencias Informáticas (UCI) asumió la tarea de desarrollar un sistema informático. Este software es capaz de gestionar el proceso de atención a la población en las instituciones cubanas, denomino Sistema de Gestión para la Atención a la Población (SIGAP). Este se concibió a través de los elementos comunes que tiene el proceso de atención a la población en los ministerios e instituciones de la administración pública en Cuba. También se tuvo en cuenta laNorma Cubana ISO 10002: Directrices para el tratamiento de las Quejas en las Organizaciones, norma en la cual se describe la metodología a seguir para desarrollar el proceso de atención a la población en Cuba.

El SIGAP permite crear un expediente único para cada persona que promueve un escrito de queja. Una vez que el escrito es registrado en el sistema y asociado a su respectivo expediente, este pasa por un grupo de estados hasta que sea cerrado. En las instituciones y ministerios donde se utilice el SIGAP, varias personas pueden interactuar con este, en correspondencia con los diferentes estados por los que puede transitar un escrito. Por tal motivo cada uno de los usuarios del sistema requiere ser notificado sobre los escritos que le sean asignados para ser procesados.

En la actualidad, el SIGAP no brinda la posibilidad de notificarle a sus usuarios sobre la asignación de escritos para ser procesados. Cada usuario debe revisar en el sistema si tiene algún escrito pendiente. Por tal motivo, si existen escritos con un corto periodo de tiempo, corren el riesgo de no ser vistos antes de su fecha de cumplimiento por la persona responsable de tramitarlo. De igual forma, el sistema no brinda la posibilidad de alertar a los usuarios sobre los escritos que se encuentran atrasados. Todo esto provoca tardanzas e incumplimientos en el área.

Por otra parte, los usuarios del sistema necesitan estar comunicados entre sí. En la actualidad esta comunicación la logran a través de vía telefónica o por correo electrónico.

Lo anterior implica que cuando los usuarios están trabajando en el SIGAP y necesitan consultar alguna información con otros usuarios, tengan que acudir a un correo electrónico o un teléfono; pues el SIGAP no brinda la posibilidad de establecer esta comunicación a través del mismo. Todo lo antes descrito provoca atrasos en el proceso de atención a la población, pues en el momento de necesitar establecerse una comunicación entre usuarios, los teléfonos pueden estar ocupados o averiados, o también se corre el riesgo de no existir un correo electrónico en la entidad donde se esté usando el sistema.

A partir de la problemática antes descrita se genera la necesidad de resolver el siguiente **problema de investigación**: ¿Cómo gestionar alertas, mensajes y notificacionesen el Sistema de Gestión para la Atención a la Población de forma tal que se contribuya a la celeridad en el proceso?

Objeto de estudio: proceso de gestión de alertas, mensajes y notificacionesen sistemas informáticos.

Objetivo general:desarrollar los componentes para la gestión de alertas, mensajes y notificaciones en el Sistema de Gestión para la Atención a la Población, de forma tal que contribuya a la celeridad en el proceso.

Campo de acción: Proceso de gestión de alertas, mensajes y notificacionesen el Sistema de Gestión de Atención a la Población.

Definiéndose como **idea a defender:** si sedesarrollan los componentes para la gestión de alertas, mensajes y notificaciones en el Sistema de Gestión para la Atención a la Población, se contribuye a la celeridad en el proceso.

Objetivos específicos:

- Elaborar el marco teórico de la investigación mediante un estudio de los referentes teóricos sobre el desarrollo de soluciones informáticas dirigidas a gestionar alertas, mensajes y notificaciones.
- Realizar la identificación de los requisitos, análisis, diseño e implementación de los componentes propuestos.
- Validar el diseño y el funcionamiento de los componentes propuestos aplicando métricas y pruebas de software respectivamente.

- Verificar el cumplimiento de la relación causa-efecto de la variable independiente sobre las variables dependientes de la investigación.

Métodos Teóricos:

Histórico-Lógico:

Permitió realizar estudios sobre las distintas tendencias en cuanto a la implementación de soluciones informáticas para la atención a la población que incluyan componentes o funcionalidades para la gestión de alertas, mensajes y notificaciones. La investigación realizada permitió conocer las características de estos sistemas, así como el momento en que fueron desarrollados, ayudando a establecer una relación cronológica entre las distintas propuestas y estudiar la evolución de estos a través del tiempo.

Analítico-Sintético:

Permitió la realización del diseño teórico de la investigación, extrayéndose los elementos más importantes relacionados con la gestión de alertas, mensajes y notificaciones, así como de las herramientas, tecnologías y buenas prácticas utilizadas en la propuesta de solución durante el análisis de la bibliografía estudiada.

Modelación:

Facilitó la representación de los distintos procesos y esquemas que intervinieron en el desarrollo de los componentes propuestos. A través de diagramas se representaron los distintos elementos del diseño, facilitando así el proceso en sí mismo al ofrecer una representación de alto nivel de cada etapa o nivel de los distintos componentes.

Métodos Empíricos:

Medición:

Permitió medir la calidad de la especificación de los requisitos y el grado de ambigüedad de estos, además de obtener una medida de la calidad del diseño para su validación.

Entrevista:

En la presente investigación se realizó una entrevista estructurada dirigida al cliente. Con el propósitodecomprender el negocio del SIGAP, así como los procesos que debían generar alertas y notificaciones. En el Anexo 2, se encuentra la guía de preguntas utilizada en el desarrollo de la entrevista.

.

El contenido de este trabajo consta de tres capítulos, definidos de la siguiente forma:

Capítulo 1: Fundamentación Teórica. En este capítulo se describen los conceptos relacionados con el objeto de estudio de la presente investigación, así como las características fundamentales de los sistemas para la gestión de atención a la población que existen en diversas regiones del mundo incluyendo Cuba. Además, se describe la metodología seleccionada para guiar el proceso de desarrollo de la presente investigación, junto a los lenguajes y herramientas utilizadas, así como los patrones de diseño.

Capítulo 2: Análisis, diseño e implementación de la solución propuesta. En este capítulo se realiza una descripción de las principales características de loscomponentes para la generación de alertas, mensajes y notificacionesen el SIGAP. En el mismo se describe el diseño de la solución propuesta, a partir de las disciplinas de la metodología definida para guiar el desarrollo de esta. Entre los contenidos analizados se encuentra el proceso de captura de los requisitos funcionales (RF) y no funcionales (RnF) con los que deben cumplir los componentes propuestos. Además, se describe la arquitectura de la solución, el diagrama de clases del diseño, el modelo de datos y los patrones de diseño utilizados. Por otra parte, se exponen los estándares de codificación empleados en la disciplina de implementación y se realiza una descripción de las principales interfaces de la solución obtenida.

Capítulo 3: Validación de la solución propuesta. En el capítulo se muestran los resultados obtenidos luego de aplicar las técnicas de validación de requisitos, así como las métricas para la validación del diseño de la solución propuesta. En cada caso se documentan los resultados obtenidos. Por otra parte, se define la estrategia de prueba aplicada a partir de las disciplinas establecidas para la etapa de pruebas por la metodología AUP (por sus siglas en inglés, *AgilUnifiedProcess*) variación UCI. En la estrategia se define realizar pruebas a nivel de unidad con la aplicación del método de caja blanca y la técnicacamino básico.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1.Introducción

En este capítulo se describen los conceptos relacionados con el objeto de estudio de la presente investigación, así como las características fundamentales de los sistemas para la gestión de atención a la población que existen en el ámbito nacional e internacional. Además, se describe la metodología seleccionada para guiar el proceso de desarrollo de la presente investigación, junto a los lenguajes y herramientas utilizadas, así como los patrones de diseño.

1.2.Definición de los principales conceptos relacionados con la investigación

Con el objetivo de lograr un entendimiento del contenido a tratar en la presente investigación, se requiere del análisis de los siguientes conceptos:

Alerta:las alertas son mensajes enviados un determinado usuario sobre eventos que van a ocurrir en un sistema y que necesitan ser informados. Esta información debe ser chequeada, en tiempo real, porque está encaminada a mejorar la situación advertida. Las mismas deben persistir en el sistema hasta tanto no se le dé solución. Estas hacen referencia a una situación de vigilancia o atención(1028-4788, 2019).

Notificación: es la acción y efecto de notificar un verbo que procede del latín y que significa comunicar formalmente una resolución o dar una noticia con propósito cierto. La noción de notificación, por lo tanto, está vinculada a una comunicación o un aviso. Al enviar una notificación, una empresa, una organización o una persona pretende dejar asentada determinada resolución que se ha tomado o que se tomará en un futuro (Gardey, 2013).

Mensaje: un mensaje es, de manera general, el objeto de comunicación que circula entre un emisor y un receptor, a través de un canal de comunicación establecido. Los sistemas digitales actuales utilizan una implementación de este concepto utilizando las tecnologías para hacer más inmediata esta comunicación, dando lugar al concepto de mensaje instantáneo. En el ámbito de las tecnologías de la información y las comunicaciones un mensaje instantáneo es una forma de comunicación en tiempo real entre dos o más personas basada en texto, el cual es enviado entre dispositivos conectados entre sí a través de una red(Bembibre, 2009).

1.3.Tendencias actuales

Los sistemas para la gestión de alertas, mensajes y notificaciones tienen como objetivo establecer la comunicación con los usuarios.Las alertas tienen la capacidad de avisar a un usuario sobrela existencia información relevante relacionada con el vencimiento de términos, procesos pendientes o la ocurrencia una acción con las que el usuario esté relacionado. Las notificaciones son generadas por la ocurrencia de eventos en un sistema o aplicación informática. Actualmente son diversos los sistemas que aplican estas funcionalidades. Ejemplos de estosen el ámbito foráneo son:

- Sistema de Gestión de quejas y denuncias ciudadanas para la alcaldía municipal de Santa Tecla situada en el Salvador: este sistema le permite a la población adjuntar fotografías que evidencien la queja reportada y capturar su posición automáticamente desde el Sistema de Posicionamiento Global (GPS, por sus siglas en inglés, *Global PositioningSystem*) del teléfono o indicándose desde un mapa *online* del municipio de Santa Tecla. Además, este sistema brinda la posibilidad de que el personal de la alcaldía pueda ubicar en un mapa digital las quejas reportadas por los ciudadanos, registrarlas, y que las posibles respuestas sean notificadas al correo electrónico del ciudadano(López de Jiménez, 2015).
- Sistema para la Gestión de quejas ciudadanas en Pastaza(Ecuador): el sistema le permite al personal de la alcaldía la posibilidad de ubicar las quejas reportadas por los ciudadanos según la gerencia asociada a dicha queja, así como la posibilidad de notificar la llegada de una nueva queja a tratar(Gallegos, 2017).
- APROWEB:es una aplicación web fácil de usar para la recepción y seguimiento de quejas. Esta solución tiene como desventaja que posee licencia propietaria. Entre sus principales funcionalidades se destacan que el sistema puede importar y notificar las quejas al sistema por medio del correo electrónico, además recuerda vía correo electrónico al personal asignado que existen acciones pendientes a realizar(APROWEB, 2020).

En cuanto a Cuba se tienen referencias de algunos sistemas como los siguientes:

- Sistema de Gestión para la Atención a la Población (SIGAP): este sistema permite crear un expediente por cada ciudadano que se dirija a una institución para presentar un escrito de queja, denuncia, solicitud o sugerencia. En este expediente se archivan todos los escritos que en lo adelante la persona presente. El SIGAP también brinda la posibilidad a los usuarios de tener un control sobre el estado de

los expedientes y escritos, así como de las respuestas recibidas desde los destinatarios a los cuales han sido dirigidos (Rojas , 2018).

- Sistema de Gestión de Quejas y Reportes del hotel “BreezesBella Costa” (SISGQR): es un sistema basado en software libre, el cual brinda la posibilidad de almacenar y notificar toda la inconformidad que pueda tener el cliente en relación con uno o varios de los servicios que brinda el hotel. Este sistema constituye la primera aplicación web cubana para la gestión de información de quejas y reportes de un hotel, que favorece una mayor organización y rendimiento en dicha área(García, Betancourt, & González , 2008).
- Sistema de Gestión de Quejas en la dirección municipal de la vivienda en Isla de la Juventud: es un sistema basado en software libre, el cual brinda entre sus principales funciones la posibilidad de almacenar las quejas y de notificar vía correo electrónico al cliente las respuestas a estas (Rodríguez, 2016)

En la siguiente tabla se realiza una comparación entre cada uno de los sistemas antes descritos, teniendo en cuenta las funcionalidades definidas para los componentes propuestos por los autores de la presente investigación:

Tabla 1. Tabla comparativa de las soluciones estudiadas.

Características	SiGAST	SiGPOS	APROWEB	SISGQR	SIGAP	SiGVIJ
Permite gestionar mensajes	No	No	No	No	No	No
Permite gestionar notificaciones	Sí	Sí	Sí	Sí	No	Sí
Permite gestionar alertas	No	No	Sí	No	No	No
Software Libre	Sí	Sí	No	Sí	Sí	Sí

Fuente:elaboración propia.

Una vez comparadas las soluciones informáticasse concluyen que todos estos sistemas carecen de las funcionalidades que dieron origen a la investigación. Por ejemplo, el sistema Aproweb cumple con la mayoría de las funcionalidades, pero es un software privativo, por tanto, incumple con la política de soberanía tecnológica que aplica Cuba. Por otra parte, Sistema de Gestión de Quejas en la dirección municipal de la vivienda en

Isla de la Juventudno posibilita la gestión de alertas y mensajes. Sin embargo, el quemás se adecúa a las características del contexto en análisis es el SIGAP, por tal motivo es necesario desarrollar los componentes propuestos para luego integrarlos al SIGAP.

1.4. Metodología de Desarrollo de Software

Las metodologías de desarrollo de software son un conjunto de procedimientos, herramientas y un soporte documental que ayuda a los desarrolladores a realizar un sistema informático(Ruiz, Palacín, & Flores , 2017).

Las metodologías se clasifican en tradicionales y ágiles. Estas últimas se basan en dos aspectos puntuales, el retrasar las decisiones y la planificación adaptativa, permitiendo potenciar aún más el desarrollo de software a gran escala. Dentro de las metodologías ágiles destaca el Proceso Unificado Ágil (AUP), la cual consiste en una versión simplificada de la metodología de desarrollo tradicional: Proceso Racional Unificado (RUP, por sus siglas en inglés, *RationalUnifiedProcess*). AUP describe la forma de desarrollar aplicaciones de software de manera fácil de entender, usando técnicas ágiles y conceptos que aún se mantienen válidos en RUP(Rodríguez S. T., 2015).

La metodología AUP propone organizar el proceso de desarrollo de software en cuatro fases (Inicio, Elaboración, Construcción, Transición). En el caso de la adaptación de esta metodología para los proyectos de la UCI se decide mantener la fase de inicio, pero modificando su objetivo, las tres restantes fases se unifican, quedando una sola denominada ejecución y se agrega una fase de cierre(Rodríguez S. T., 2015). A continuación de describen cada unas de estas tres fases:

Inicio: durante el inicio del proyecto se llevan a cabo las actividades relacionadas con la planeación de este. En esta fase se realiza un estudio inicial de la organización cliente, obteniéndose información acerca del alcance del proyecto, estimaciones de tiempo, esfuerzo y costo. Todo este estudio permite decidir si se ejecuta o no el proyecto(Rodríguez S. T., 2015).

Ejecución: en esta fase se ejecutan las actividades requeridas para desarrollar el software, incluyendo el ajuste de los planes del proyecto. Durante el desarrollo se modela el negocio, obtienen los requisitos, se elaboran la arquitectura y el diseño. Además, se implementa el software y se le aplican pruebas. Durante esta fase el producto es transferido al ambiente de los usuarios finales o entregado al cliente. Asimismo, en la

transición se capacita a los usuarios finales sobre la utilización del software (Rodríguez S. T., 2015).

Cierre: en esta fase se analizan los resultados del proyecto relacionados con su ejecución y se realizan las actividades formales de cierre del proyecto(Rodríguez S. T., 2015).

Disciplinas de variación de AUP-UCI

AUP propone siete disciplinas (Modelo, Implementación, Prueba, Despliegue, Gestión de configuración, Gestión de proyecto y Entorno). Para el ciclo de vida de los proyectos de la UCI se decide utilizar igual número de disciplinas, pero a un nivel más atómico. Los flujos de trabajos: modelado de negocio, requisitos y análisis y diseño en AUP están unidos en la disciplina modelo. En la variación AUP para la UCI estos flujos de trabajo se consideran disciplinas independientes, además se mantiene la disciplina implementación. En el caso de las pruebas, estas se desagregan en tres disciplinas: pruebas internas, pruebas de liberación y pruebas de aceptación. Las disciplinas de AUP asociadas a la gestión de proyecto, en la variación UCI se cubren con las áreas de procesos que define CMMI-DEV v1.3 para el nivel 2 (gestión de la configuración, planeación de proyecto y monitoreo y control)(Rodríguez S. T., 2015).

Escenarios para la disciplina Requisitos

A partir de que el Modelado de negocio propone tres variantes a utilizar en los proyectos (Caso de Uso del Negocio (CUN), Diagrama de Proceso del Negocio (DPN) y Modelo Conceptual (MC)). Existen tres formas de encapsular los requisitos (Caso de Uso del Sistema (CUS), Historias de Usuarios (HU), Diagrama de Requisitos por Proceso (DRP)), surgen cuatro escenarios para modelar el sistema en los proyectos, manteniendo en dos de ellos el MC, quedando de la siguiente forma:

- Escenario No 1: proyectos que modelen el negocio con CUN solo pueden modelar el sistema con CUS.

  Se aplica a los proyectos que hayan evaluado el negocio a informatizar y como resultado obtengan que puedan modelar una serie de interacciones entre los trabajadores del negocio/actores del sistema (usuario), similar a una llamada y respuesta respectivamente, donde la atención se centra en cómo el usuario va a utilizar el sistema. Es necesario que se tenga claro por el proyecto que los CUN muestran como los procesos son llevados a cabo por personas y los activos de la organización(Rodríguez S. T., 2015).

- Escenario No 2: proyectos que modelen el negocio con MC solo pueden modelar el sistema con CUS.

  Se aplica a los proyectos que hayan evaluado el negocio a informatizar y como resultado obtengan que no sea necesario incluir las responsabilidades de las personas que ejecutan las actividades, de esta forma modelarían exclusivamente los conceptos fundamentales del negocio. Se recomienda este escenario para proyectos donde el objetivo primario es la gestión y presentación de información(Rodríguez S. T., 2015).
- Escenario No 3:proyectos que modelen el negocio con DPN solo pueden modelar el sistema con DRP.

  Se aplica a los proyectos que hayan evaluado el negocio a informatizar y como resultado obtengan un negocio con procesos muy complejos, independientes de las personas que los manejan y ejecutan, proporcionando objetividad, solidez, y su continuidad. Se debe tener presente que este escenario es muy conveniente si se desea representar una gran cantidad de niveles de detalles y la relaciones entre los procesos identificados(Rodríguez S. T., 2015).
- Escenario No 4: proyectos que no modelen negocio solo pueden modelar el sistema con HU.

  Se aplica a los proyectos que hayan evaluado el negocio a informatizar y como resultado obtengan un negocio muy bien definido. El cliente estará siempre acompañando al equipo de desarrollo para convenir los detalles de los requisitos y así poder implementarlos, probarlos y validarlos. Se recomienda en proyectos no muy extensos, ya que una HU no debe poseer demasiada información(Rodríguez S. T., 2015).

La aplicación de la variación de AUP en la UCI permite estandarizar el proceso de desarrollo de software en la universidad, dando cumplimiento a las buenas prácticas que define CMMI-DEV v1.3. Estas disciplinas se desarrollan en la fase de ejecución y pueden estar organizadas en iteraciones, con el propósito de obtener resultados incrementales.

A partir del análisis realizado sobre la variación de la metodología AUP para la UCI, los autores de la presente investigación deciden organizar el proceso de desarrollo de los componentes propuestos a partir de las fases y disciplinas definidas en esta variación de la metodología. Esta decisión es adoptada teniendo en cuenta quela solución propuesta

forma parte de uno de los proyectos de la red de centros productivos de la universidad, red que tiene definido el uso de la metodología AUP para la UCI.

El escenario a utilizar es el No.4, ya que aplica a los proyectos que hayan evaluado el negocio a informatizar y como resultado obtengan un negocio muy bien definido. Estas características se adaptan a la solución propuesta.

1.5 Herramientas de Modelado

Las herramientas CASE (por sus siglas en inglés, *ComputerAided Software Engineering*)están destinadas a aumentar la productividad en el desarrollo del software reduciendo el coste del mismo en términos de tiempo y de dinero. Estas herramientas pueden ayudar en todos los aspectos del ciclo de vida de desarrollo del software en tareas como el diseño de proyectos, cálculo de costos, implementación de parte del código a partir del diseño dado, compilación automática y documentación o detección de errores (Visual-Paradigm, 2019).

A continuación, se describe la herramienta CASE utilizada en las tareas ingenieriles de la presente investigación.

1.5.1.Visual Paradigm 8.0

Es una herramienta UML(por sus siglas en inglés, *UnifiedModelingLanguage*) profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Permite modelar todos los tipos de diagramas de clases, generar código desde diagramas y generar documentación. La herramienta agiliza la construcción de aplicaciones con calidad y a un menor coste de tiempo. Posibilita la generación de bases de datos, transformación de diagramas de Entidad-Relación en tablas de base de datos, así como obtener ingeniería inversa de bases de datos (Visual Paradigm, 2020).

1.6.Lenguajes de programación

De los lenguajes de programación que existen para desarrollar aplicaciones orientadas a la web se encuentran dos grupos fundamentales, la programación del lado del servidor y la programación del lado del cliente. Esta última incluye aquellos lenguajes que son interpretados por una aplicación cliente como el navegador web, entre ellos se encuentra HTML (por sus siglas en inglés, *HyperTextMarkupLanguage*). Los lenguajes de

programación del lado del servidor son reconocidos, ejecutados e interpretados por el propio servidor, el que se encarga de brindar información al cliente en un formato comprensible para él. Para el desarrollo de la solución propuesta se hace necesario utilizar los lenguajes que a continuación se describen, teniendo en cuenta que responden a las tecnologías y lenguajes en que se desarrolla el sistema.

1.6.1. Java SE11

Java (desarrollado por *Sun Microsystems* y posteriormente adquirida por la compañía *Oracle*) es un lenguaje de programación de propósito general, concurrente, orientado a objetos y multiplataforma que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. El mismo permite desarrollar e implementar aplicaciones *Java* en equipos de escritorio y servidores. Proporciona una colección de clases para su uso en aplicaciones de red, que permite establecer y aceptar conexiones con servidores o clientes remotos, facilitando así la creación de aplicaciones distribuidas (Oracle, 2020).

1.6.2. JavaScript ES6

Es un lenguaje de programación ligero, interpretado por la mayoría de los navegadores y que les proporciona a las páginas web, efectos y funciones complementarias a las consideradas como estándar HTML. Este tipo de lenguaje de programación es de código abierto, por lo que cualquier persona puede utilizarlo sin comprar una licencia. Con frecuencia son empleados en los sitios web, para realizar acciones en el lado del cliente, estando centrado en el código fuente de la página web (Mozilla, 2020).

1.6.3.TypeScript 2.9

Es un lenguaje de programación de código abierto desarrollado por Microsoft, el cual cuenta con herramientas de programación orientada a objetos, muy favorable si se tienen proyectos grandes. *Typescript* convierte su código en *Javascript* común. Es llamado también *superset* de *Javascript*, lo que significa que, si el navegador está basado en *Javascript*, este nunca llegará a comprender que el código original fue realizado con *Typescript* y ejecutará el *Javascript* como lenguaje original(Typescript, Microsoft, 2019).

1.6.4. HTML 5.0

HTML es el lenguaje que se utiliza para crear las páginas web de internet. Es reconocido universalmente y permite publicar información de forma global. Define una estructura básica y un código HTML para la definición de contenido de una página web, como texto,

imágenes, entre otros y se basa en la referenciación por hipertextos o enlaces entre páginas(Mozilla, 2019).

1.6.5. CSS 3.0

CSS (por sus siglas en inglés, *Cascading Style Sheets*) es un lenguaje para controlar la presentación de los documentos electrónicos definidos con HTML y XHTML(por sus siglas en inglés*, ExtensibleHyperTextMarkupLanguage*). CSS es la mejor forma de separar los contenidos de su presentación y es imprescindible para la creación de páginas web complejas.Mejora la accesibilidad del documento, reduce la complejidad de su mantenimiento y permite visualizar el mismo documento en infinidad de dispositivos diferentes.Se utiliza para definir el aspecto de todos los contenidos, como: el color, tamaño y tipo de letra de los párrafos de texto, la separación entre titulares y párrafos, la tabulación con la que se muestran los elementos de una lista. Esta forma de descripción de estilos ofrece a los desarrolladores el control total sobre estilo y formato de sus documentos. Funciona a base de reglas para las declaraciones sobre el estilo de uno o más elementos(Mozilla, 2019).

1.7.Marcos de trabajo

Desde el punto de vista del desarrollo de software, un marco de trabajo es una estructura de soporte definida, en la cual otro proyecto de software puede ser organizado y desarrollado (Alegsa, 2020).

1.7.1. Angular 8

Es un marco de trabajo de JavaScript y TypeScript de código abierto desarrollado por Google**.** Contiene un conjunto de librerías útiles para el desarrollo de aplicaciones web y propone una serie de patrones de diseño para levarlas a cabo. Además, contiene todas las herramientas que los creadores ofrecen para que los desarrolladores sean capaces de crear un HTML enriquecido. La palabra clave que permite dicho HTML enriquecido en Angular es "directiva", que no es otra cosa que código JavaScript que mejora el HTML. También promueve y usa patrones de diseño de software básicamente en el patrón Modelo Vista Controlador(Angular, 2020).

1.7.2. Spring Boot 1.5.6

Es un sub-proyecto de Spring, el mismo facilita la creación de proyectos con el marco de trabajo Spring, eliminando la necesidad de crear largos archivos de configuración XML

(por sus siglas en inglés,*Extensible MarkupLanguage*). Además, provee configuraciones por defecto para Spring y otras librerías, también provee un modelo de programación parecido a las aplicaciones java tradicionales que se inician en el método *main*(Spring, 2020).

1.7.3.Bootstrap 4.3.1

Es una librería multiplataforma dirigida al diseño y desarrollo de interfaces gráficas del lado del cliente, de sitios y aplicaciones web. Contiene plantillas de diseño predefinidas con diversos elementos como botones, formularios, cuadros,menús de navegación y otros elementos basados en HTML y CSS.También contiene extensiones adicionales de *Javascript*(Bootstrap, 2020).

Los autores de la presente investigación deciden utilizar estos marcos de trabajo con el propósito de lograr la compatibilidad del componente con el SIGAP. Sistema que también está desarrollado sobre esta tecnología.

1.8.Entornos de Desarrollos Integrados

Un IDE(por sus siglas en inglés, *IntegratedDevelopmentEnvironment*), es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica. Los IDE proveen un marco de trabajo amigable para la mayoría de los lenguajes de programación (Martin Olivera & Zamora Sanchez, 2016).

Para el desarrollo de la solución propuesta, se utilizan como IDE las herramientas *IntelliJ IDEA*en su versión 2019.2 para la comunidad y *WebStorm* en su versión 2019.2, teniendo en cuenta que ambas son libres, de código abierto y se integran perfectamente alos lenguajes de programación a utilizar. Además, el componente propuesto debe ser integrado sobre una solución que también está implementada sobre estos entornos de desarrollo.

1.8.1. IntelliJ IDEA 2019.2

Es un IDE desarrollado por *JetBrains* y está disponible en dos ediciones: una edición para la comunidad (código abierto) y una edición comercial (de pago). Es soportado por varios sistemas operativos: *Windows, Linux* o Mac OS. Su versión gratuita soporta lenguajes como *Java, Groovy, XML/XSL, Kotlin y Python, Clojure, Dart, Erlang, Go, Haxe, Perl, Scala, Haskell, Lua*, estos últimos vía *plugin*. La versión de pago además de soportar

todos los lenguajes antes mencionados soporta:*JavaScript*, HTML, CSS, SQL, Ruby, así como PHP vía *plugin*(JetBrains, 2020).

1.8.2. WebStorm 2019.2

Es un entorno de desarrollo inteligente, entiende el proyecto y ayuda a producir código de alta calidad de manera más eficiente, gracias al completamiento de código, se detectan errores sobre la marcha, la navegación y refactorizaciones automatizadas. Da soporte a las recientes tecnologías, funcionando de la mejor manera con las más modernas y populares para el desarrollo web, así como *AngularJS*, *ECMAScript 6* y *Compass*. Es multiplataforma, funciona en *Windows*, *MacOS* o *Linux* con una única clave de licencia. *WebStorm* agiliza el flujo de trabajo mediante la integración con todo lo necesario para el desarrollo productivo. Además, desde el IDE se pueden utilizar varias herramientas como el depurador y terminales (JetBrains, 2020).

1.9.Sistema Gestor de Bases de Datos

Un Sistema Gestor de Bases de Datos (SGBD) es un conjunto de programas no visibles que administran y gestionan la información que contiene una base de datos. A través de él se maneja todo acceso a la base de datos con el objetivo de servir de interfaz entre ésta, el usuario y las aplicaciones**(Rouse M. , TeachTarget, 2019)**.Para el desarrollo de la solución propuesta se hace necesario utilizar el PostgreSQL como sistema gestor de bases de datos teniendo en cuenta la compatibilidad de este con el software SIGAP.

1.9.1. PostgreSQL 9.6

Es un potente sistema de base de datos objeto-relacional de código abierto. Cuenta con más de 15 años de desarrollo activo y una arquitectura probada con una alta fiabilidad e integridad de datos. Se ejecuta en los principales sistemas operativos que existen en la actualidad como *Linux*, UNIX (AIX, BSD, HP-UX, SGI IRIX, *MacOSX*, *Solaris*, *Tru64*) y *Windows*(PostgreSQL, 2020).

1.10.Otras tecnologías

Para el desarrollo de la solución propuesta fue necesario utilizar otras tecnologías como el caso de Kafka. Apache Kafka es una plataforma distribuida de transmisión continua o sistema de registro de confirmación continua. Está construido como un sistema de mensajería bajo el modelo Publicador/Subscriptor, ofreciendo una API (por sus siglas en inglés, *AplicationProgramming Interface*) limpia y concisa para implementar sistemas

distribuidos. Su diseño está orientado a resolver principalmente los problemas de complejidad e integración que presentan otras plataformas similares como *Hadoop* o *Spark*, siendo sus principales características la escalabilidad, durabilidad y tolerancia a fallos de su diseño. Apache Kafka permite almacenar en el sistema de archivos un registro inmutable y permanente de cada operación que se realice durante la ejecución de la aplicación, siendo ideal para la programación de arquitecturas orientadas a eventos(Stopford, 2018).

1.11.Patrones de diseño

Los patrones de diseños son un conjunto de soluciones a problemas de diseño que ocurren una y otra vez en el desarrollo de software. Funcionan como una plantilla de solución donde una solución abstracta es adaptada por el usuario a su problema. Además, los patrones de diseño proveen un idioma común entre distintos lenguajes de programación que permite a arquitectos y desarrolladores comunicarse de manera común y abordar problemas sin importar el lenguaje de programación utilizado (Rocha, 2018).

Objetivos de estospatrones**:**

- Proporcionar catálogos de elementos reusables en el diseño de sistemas software.
- Evitar la reiteración en la búsqueda de soluciones a problemas ya conocidos y solucionados anteriormente.
- Formalizar un vocabulario común entre diseñadores.
- Estandarizar el modo en que se realiza el diseño.
- Facilitar el aprendizaje de las nuevas generaciones de diseñadores condensando conocimiento ya existente.

Así mismo, no pretenden:

- Imponer ciertas alternativas de diseño frente a otras.
- Eliminar la creatividad inherente al proceso de diseño.
- No es obligatorio utilizar los patrones siempre, sólo en el caso de tener el mismo problema o similar que soluciona el patrón, siempre teniendo en cuenta que en un caso particular puede no ser aplicable. Abusar o forzar el uso de los patrones puede ser un error (Rouse M. , Tech Target, 2020).

En el capítulo 2 los autores describen el uso de los patrones utilizados en el diseño e implementación de los componentes propuestos.

1.12.Pruebas

Uno de los elementos principales para certificar la calidad de una aplicación informática lo constituye el resultado de las pruebas que se practican. Un control y una gestión de calidad implementados de forma adecuada, especialmente durante el proceso de desarrollo del software, aumentan la calidad del sistema final, reducen los costos de avance y acortan el tiempo necesario para el desarrollo. Para determinar el nivel de calidad se deben efectuar medidas o pruebas que permitan comprobar el grado de cumplimiento respecto a las especificaciones principales del sistema(Pressman R. , 2010). Teniendo en cuenta lo anterior a continuación de define el concepto de pruebas de software.

Pruebas de software: comprenden el conjunto de actividades que se realizan para identificar posibles fallos de funcionamiento, configuración o usabilidad de un programa o aplicación, por medio de pruebas sobre el comportamiento del mismo(Pressman R. , 2010).

Las pruebas de software se caracterizan por niveles o etapas**.**A continuación, se muestra una breve descripción de cada nivel o etapade prueba:

- Pruebas modulares o a nivel de unidad: este tipo de pruebas son ejecutadas normalmente por el equipo de desarrollo, consisten en la ejecución de actividades que le permitan verificar al desarrollador que los componentes unitarios están codificados bajo condiciones de robustez, esto es, soportando el ingreso de datos erróneos o inesperados y demostrando así la capacidad de tratar errores de manera controlada. Adicionalmente, las pruebas sobre componentes unitarios, suelen denominarse pruebas de módulos o pruebas de clases, siendo la convención definida.
- Pruebas de Integración:este tipo de pruebas son ejecutas por el equipo de desarrollo y consisten en la comprobación de que elementos del software que interactúan entre sí, funcionan de manera correcta (Londoño, 2005).
- Pruebas de Sistema:este tipo de pruebas deben ser ejecutadas idealmente por un equipo de pruebas ajeno al equipo de desarrollo, una buena práctica en este punto corresponde a la tercerización de esta responsabilidad. La obligación de este equipo, consiste en la ejecución de actividades de prueba en donde se debe

verificar que la funcionalidad total de un sistema fue implementada de acuerdo a los documentos de especificación definidos en el proyecto. Los casos de prueba a diseñar en este nivel de pruebas, deben cubrir los aspectos funcionales y no funcionales del sistema. Para el diseño de los casos de prueba en este nivel, el equipo debe utilizar como bases de prueba entregables tales como: requerimientos iniciales, casos de uso, HU, diseños, manuales técnicos y de usuario final (Londoño Abad, 2005).

- Pruebas de Aceptación:independientemente de que se haya tercerizado el proceso de pruebas y así la firma responsable de estas actividades haya emitido un certificado de calidad sobre el sistema objeto de prueba, es indispensable, que el cliente designe a personal que haga parte de los procesos de negocio para la ejecución de pruebas de aceptación, es incluso recomendable, que los usuarios finales que participen en este proceso, sean independientes al personal que apoyó el proceso de desarrollo (Londoño Abad, 2005).

Teniendo en cuenta la metodología seleccionada para guiar el desarrollo del componente propuesto por la presente investigación, a continuación, se describen las disciplinas de pruebas de software que esta plantea:

- Pruebas Internas:son realizadas por sus propios desarrolladores, verificando el resultado de la implementación, probando cada construcción según sea necesario, así como las versiones finales a ser liberadas. Los artefactos necesarios para la realización de estas pruebas son los casos de pruebas (Rodríguez Sanchéz, 2015).
- Pruebas de Liberación:son diseñadas y ejecutadas por una entidad certificadora de la calidad externa, a todos los entregables de los proyectos antes de ser entregados al cliente para su aceptación (Rodríguez Sanchéz, 2015).
- Pruebas de Aceptación: son las únicas pruebas que son realizadas por los clientes. Consiste en comprobar si el producto está listo para ser implantado para el uso operativo en el entorno del usuario(Rodríguez Sanchéz, 2015).

Para el desarrollo de la solución propuesta se adopta como estrategia de prueba aplicar los cuatro niveles y realizando pruebas funcionales con las técnicas que estas incluyen. En el capítulo 3 se realiza una descripción más amplia de la estrategia de prueba definida por los autores de la presente investigación.

1.13.Conclusiones parciales

El estudio de los conceptos asociados a los términos de alerta, notificaciones y mensajes, propician un mejor entendimiento del objeto de estudio de la presente investigación. Por otra parte, el uso de métodos científicos, tanto teóricos como empíricos fue fundamental para la profundización del contenido del presente trabajo. El estudio de sistemas homólogos de gestión de atención a la población, tanto nacionales como internacionales, constituyó un importante punto de referencia para conocer en mayor detalle las características de estos sistemas. También se contribuyó a la definición de algunas funcionalidades delos componentes propuestos y se demostró la necesidad de desarrollar la solución propuesta.La decisión de utilizar la metodología AUP para la UCI en su escenario número 4,así como el mismoentorno de desarrolloutilizado en el proyecto XABAL SIGAP 1.0 al cual responde la presente investigación, permite que los autores de los componentes propuestos disminuyan la curva de aprendizaje en el trabajo con la metodología, tecnologías y herramientas empleadas.

CAPÍTULO 2: ANÁLISIS, DISEÑO E IMPLEMENTACIÓN DE LA SOLUCIÓN PROPUESTA

2.1. Introducción

Este capítulo provee una descripción general de la propuesta de solución. Se realiza la definición de RF y RnF, validándose cada uno de estos mediante la aplicación de técnicas como: creación de prototipos, generación de casos de prueba y revisiones formales de los requisitos. Igualmente se describen las HU. En la etapa de análisis y diseño se modela y caracteriza la arquitectura de los componentes propuestos, a través de los diagramas correspondientes, especificándose cada uno de los componentes que la integran y cómo estos interactúan entre sí. De igual manera se describen y ejemplifican los patrones de diseño utilizados. Por último, se definen los estándares de codificación empleados durante la implementación de la solución.

2.2. Descripción del negocio

El SIGAP surge en el año 2018 como resultado de un acuerdo de colaboración entre la UCI y la Asamblea Nacional del Poder Popular, proyecto asumido por un equipo de desarrolladores del CEGEL. Una vez concluido su desarrollo, se decidió convertirlo en un producto y otras instituciones cubanas como la Contraloría General de la República, el Ministerio de Comercio Interior y el Comité Central del Partido Comunista de Cuba se han interesado por la utilización de esta solución informática encargada de gestionar el proceso de atención a la población en una institución.

Este software permite crear un expediente único para cada persona que promueve un escrito de queja, denuncia o sugerencia. Una vez que el escrito es registrado en el sistema y asociado a su respectivo expediente, este pasa por un grupo de estados hasta ser cerrado, lo cual garantiza un correcto seguimiento del proceso de atención a la población.

Con el propósito de lograr un cumplimiento adecuado de las tareas realizadas en el sistema, los usuarios necesitan ser notificados acerca de eventos que tienen lugar en el mismo, como la asignación al usuario de escritos para ser procesados. De igual manera necesitan ser alertados acerca de eventos de mayor prioridad como escritos que se encuentran atrasados por estar pasados del término reglamentado. Por otra parte, los usuarios del sistema necesitan estar comunicados entre sí, lo cual da lugar a la necesidad de poder enviar mensajes a otros usuarios a través de sistema y de manera eficiente.

Una notificación puede ser enviada a un usuario por diversas razones, siendo algunas de las más importantes la asignación de un escrito para ser procesado, la recepción de un mensaje enviado por otro usuario, el cambio en el estado de un escrito, o la reasignación de un escrito para ser procesado. De esta manera el usuario recibe información en tiempo real de la ocurrencia de tales eventos en el sistema.

De manera similar una alerta es generada ante la ocurrencia de eventos, ejemplos: el vencimiento del término reglamentado, la proximidad de un escrito al fin del plazo establecido para ser procesado oel vencimiento del plazo establecido para darle cumplimiento a una tarea. En general cualquier suceso en el sistema que demande una acción inmediata por parte del usuario.

Por otro lado, el envío de mensajes es una funcionalidad presente en muchos sistemas. En el caso de la solución propuesta, cada usuario puede comunicarse directamente con el resto de los usuarios a través del sistema, enviando mensajes en formato texto a uno o más usuarios.

Teniendo en cuenta el negocio antes descrito, a continuación, se describen los requisitos funcionales y no funcionales definidospara el desarrollo de los componentes para la gestión de alertas, mensajes y notificaciones en el Sistema de Gestión para la Atención a la Población.

2.3. Requisitos

La tarea principal de la disciplina Requisitos es desarrollar un modelo del sistema que se va a construir. Esta disciplina comprende la administración y gestión de los requisitos funcionales y no funcionales del producto(Rodríguez Sanchéz, 2015).

En la ingeniería de software, un requisito es una necesidad documentada sobre el contenido, forma o funcionalidad de un producto o servicio, o sea, una condición o capacidad que debe ser conformada por el sistema. En la ingeniería clásica, estos se utilizan como datos de entrada en la etapa de diseño del producto(Pressman R. , 2010).

A continuación, se describen las técnicas empleadas en la obtención de los requisitos de la propuesta de solución:

- Tormenta de ideas:es una técnica de reuniones en grupo cuyo objetivo es la generación de ideas en un ambiente libre de críticas o juicio(Pressman R. , 2010). Esta técnica se evidencia durante las reuniones con el cliente donde, luego de un diálogo entre ambas partes, se obtuvo como resultado un conjunto de acuerdos con el fin de refinar las

necesidades del cliente. En el Anexo 1 del presente documento se relacionan los acuerdos tomados durante la tormenta de ideas.

- Entrevista:la entrevista es de gran utilidad para obtener información cualitativa como opiniones, o descripciones subjetivas de actividades. Es una técnica muy utilizada, y requiere una mayor preparación y experiencia por parte del analista(Sommerville I. , 2011). Esta técnica se evidencia durante el levantamiento de información realizado en las reuniones realizadas con el cliente, donde se formularon un conjunto de preguntas con el objetivo de entender las necesidades del mismo y concebir un lenguaje común entre el cliente y el equipo de desarrollo de la solución propuesta. En el Anexo 2 del presente documento se muestra la entrevista que se le aplicó al cliente.

2.3.1. Requisitos funcionales

Los requisitos funcionales (RF) de un sistema, son aquellos que describen cualquier actividad que este deba realizar, en otras palabras, el comportamiento o función particular de un sistema o software cuando se cumplen ciertas condiciones. Por lo general, estos deben incluir funciones desempeñadas por pantallas específicas, descripciones de los flujos de trabajo a ser desempeñados por el sistema y otros requerimientos de negocio(Pressman R. , 2010).

Luego de obtener la información necesaria e identificar las necesidades del cliente, se definieron un total de 16 requisitos funcionales. En la Tabla 2 se describen cada uno de los requisitos funcionales definidos.

Tabla 2. Descripción de los requisitos identificados.

No.	Nombre	Descripción
RF1	Generar alerta	Permite que el sistema sea capaz de observar indicadores y si sobrepasan sus límites lanzar una alerta.
RF2	Cambiar estado de la alerta	Permite leer una alerta y cambiar del estado de no leída al estado de leída.
RF3	Listar alerta	Permite mostrar un listado de las alertas correspondientes al usuario autenticadoen el sistema.
RF4	Eliminar alerta	Permite eliminar una alerta recibida, removiéndola del panel de alertas.
RF5	Generar notificación	Permite notificar a los implicados cuando sucede un evento en el sistema.

RF6	Cambiar estado de la notificación	Permite leer una notificación y cambiar del estado de no leída al estado de leída.
RF7	Listar notificación	Permite mostrar un listado de las notificacionescorrespondientes al usuario autenticado en el sistema.
RF8	Eliminar notificación	Permite eliminar una alerta recibida, removiéndola del panel de alertas.
RF9	Enviar mensaje	Permite enviar un mensaje a un usuario en el sistema.
RF10	Cambiar estado del mensaje	Permite leer un mensaje y cambiar del estado de no leído al estado de leído.
RF11	Eliminar mensaje	Permite eliminar un mensaje.
RF12	Reenviar mensaje	Permite que un mensaje se pueda reenviar a otro usuario.
RF13	Listar mensaje	Permite mostrar un listado de los mensajes correspondientes al usuario autenticado en el sistema.
RF14	Crear conversación	Permite crear una conversación entre dos usuarios.
RF15	Listar conversación	Permite mostrar un listado de las conversaciones correspondientes al usuario autenticado en el sistema.
RF16	Eliminar conversación	Permite eliminar una conversación.

Fuente: elaboración propia.

2.3.2. Requisitos no funcionales

Los requisitos no funcionales (RnF) representan características generales y restricciones de la aplicación o sistema que se esté desarrollando. Suelen presentar dificultades en su definición dado que su conformidad o no conformidad podría ser sujeto de libre interpretación, por lo cual es recomendable acompañar su definición con criterios de aceptación que se puedan medir(Pressman R. , 2010). A continuación, se listan los requisitos no funcionales definidos para el desarrollo de la solución propuesta:

Usabilidad

RnF1:los mensajes visualizados en el sistema deben ser en idioma español.

RnF2: la tipografía debe ser uniforme, de un tamaño adecuado.

RnF3: el sistema debe ofrecer una interfaz con un diseño sencillo y amigable, fácil de operar.

Eficiencia

RnF4. Requisito de eficiencia 1

Los procesos del sistema que se implementan con transacciones donde se modifica la base de datos, deben tener tiempos de respuesta no mayores a los 5 segundos. En el caso de la obtención de información a través de transacciones que involucran consultas a la base de datos, el tiempo está dado por el volumen de la misma, por lo cual se implementará en todo momento buscando simplificar, al máximo, los datos que se consulten, de manera que la cifra no exceda los 30 segundos.

Restricciones del diseño y la implementación

RnF5:la PC cliente del sistema debe poseer resolución de 1024 por 768 pixeles o superior.

RnF6: el sistema se debe desarrollar utilizando el marco de trabajo del lado del servidor: *Spring Boot* en su versión 1.5.6 o superior, mientras que como marco de trabajo del lado del cliente se debe usar Angular en su versión 5.2 o superior, además se debe hace uso de otras tecnologías como HTML5, CSS, *Typescript* y *Javascript*.

Interfaz de usuario

RnF7:las listas se mostrarán en tablas, las mismas se paginarán de 10 elementos por páginas siempre del lado del servidor, las acciones en lotes y demás acciones generales sobre los elementos de la misma, irán ubicadas al lado superior izquierdo de la tabla, las acciones individuales sobre los elementos se ubicarán en cada una de las filas, siempre al final, en la última columna, en forma de botones.

RnF8:los botones tienen dos tonalidades de colores, los de acciones primarias, que son aquellas acciones que terminan o completan una acción, son los que llamen la atención a la vista y los secundarios, que son las acciones que puede hacer el usuario entre el inicio y fin de una acción, son de un color más claro.

2.3.3. Validación de los requisitos

La validación de requisitos examina las especificaciones para asegurar que todos los requisitos del sistema han sido establecidos sin ambigüedad, sin inconsistencias, sin omisiones, que los errores detectados hayan sido corregidos, y que el resultado del trabajo se ajusta a los estándares establecidos para el proceso, el proyecto y el producto (Pressman R. , 2010).

Para la validación de los requisitos se tuvieron en cuenta las técnicas de:

- Creación de prototipos: en esta aproximación a la validación, se muestra un modelo ejecutable del sistema en cuestión a los usuarios finales y clientes. Así, ellos podrán experimentar con este modelo para constatar si cubre sus necesidades reales(Sommerville I. , 2011). En la presente investigación se realizó un modelo ejecutable utilizando el componente para que los clientes interactuaran con él y así poder determinar si cumple sus necesidades.
- Generación de casos de prueba: los requerimientos deben ser comprobables. Si las pruebas para los requerimientos se diseñan como parte del proceso de validación, esto revela con frecuencia problemas en los requerimientos. Si una prueba es difícil o imposible de diseñar, esto generalmente significa que los requerimientos serán difíciles de implementar, por lo que deberían reconsiderarse. El desarrollo de pruebas a partir de los requerimientos del usuario antes de escribir cualquier código es una pieza integral de la programación extrema (Sommerville I. , 2011).
- Revisiones formales de los requisitos: se realizaron revisiones de cada requisito, comprobando que los mismos no presenten errores e inconsistencia. Esta técnica fue ejecutada por un equipo de personas, validando la homogeneidad en la interpretación de cada una de las descripciones, y que no presenten errores u omisiones, permitiendo así evaluar la calidad de la especificaciónde cada uno de los miembros del equipo.

Para medir la calidad de la especificación de los requisitos de software se aplicó la métrica Calidad de la especificación (CE). El empleo de esta métrica permite obtener un alto nivel de entendimiento y precisión de los requisitos, para llegar a ello, se debe primeramente calcular el total de requisitos de software como se muestra a continuación:

Nr: total de requisitos de software.

Nf: cantidad de requisitos funcionales.

Nnf: cantidad de requisitos no funcionales.

Nr = Nf + Nnf

Sustituyendo los valores en la ecuación, se obtiene:

Nr = 16+ 8

Nr = 24

Finalmente, para calcular la especificidad de los requisitos (ER) se tuvo en cuenta la ausencia de ambigüedad en los mismos de forma tal que se obtuviera una sola interpretación, lo que requiere que cada uno sea descrito utilizando un término único, para ello se realiza la siguiente operación:

Nui: número de requisitos para los cuales todos los revisores tuvieron interpretaciones idénticas.

$$Q1 = Nui / Nr$$

Para sustituir los valores de las variables en la ecuación se tuvo en cuenta que, todos los requisitos se obtuvieron a través de interpretaciones idénticas del problema. Por tanto, la variable Q1 obtiene el siguiente valor:

$$Q1 = 24/ 24$$

$$Q1 = 1$$

Es importante aclarar que mientras más cerca de 1 está el valor de Q1, menor es la ambigüedad. Teniendo en cuenta el resultado anterior, igual a 1, se concluye que el 100% de los requisitos se obtuvieron a través de una sola interpretación.

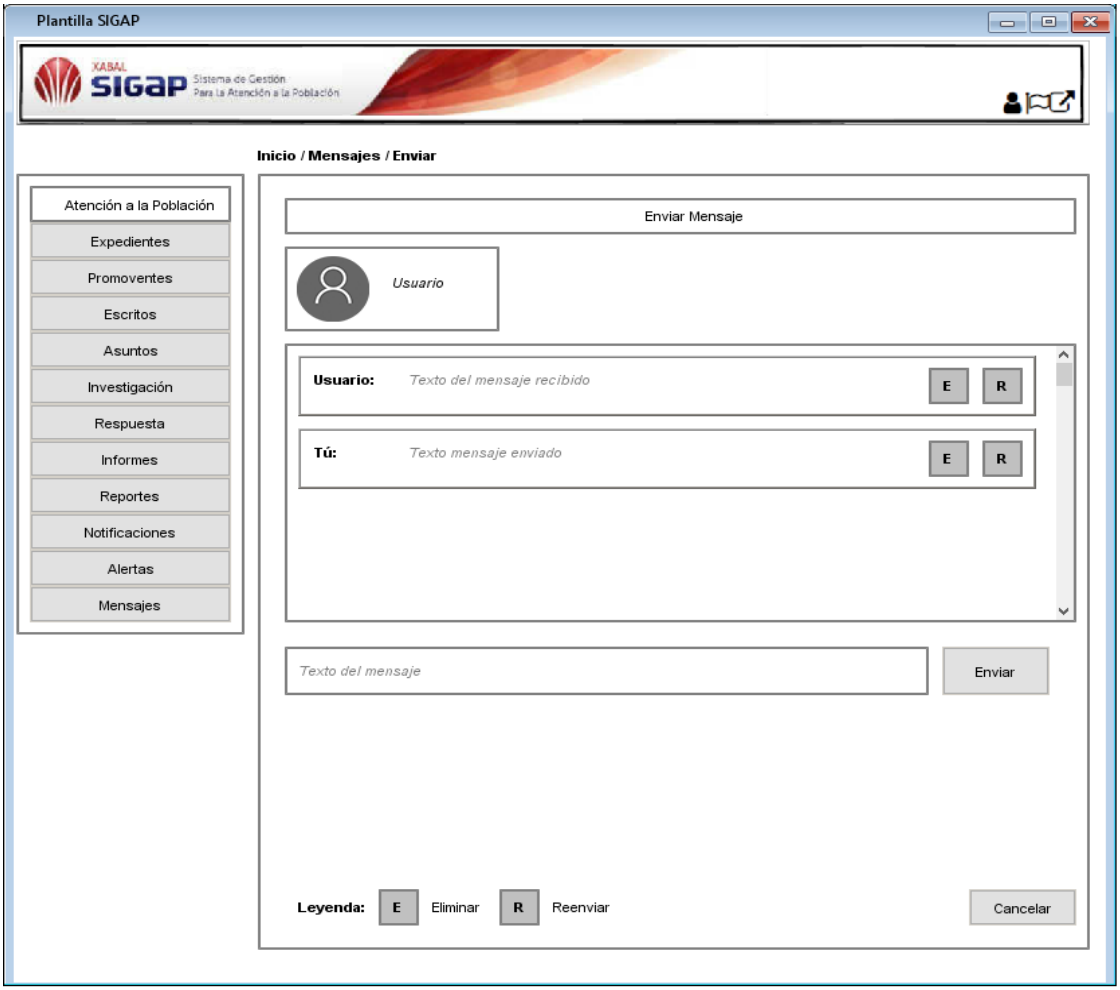
2.4. Historias de usuario

Las HU son descripciones, siempre muy cortas y esquemáticas, que resumen la necesidad concreta al desarrollar un producto o servicio. Es por eso que se conocen, además, como una representación de un requisito escrito en una o dos frases utilizando el lenguaje común del usuario(Solving Ad Hoc, 2017).

Para el desarrollo de la solución propuesta se diseñaron 14HU. A continuación, en la Tabla 3 se describe la HU “Enviar mensaje”, de prioridad alta, teniendo en cuenta el impacto que tiene en el desarrollo de la solución propuesta. En el Anexo 3, se pueden consultar el resto de las HU que responden en los RF de la presente investigación.

Tabla 3. Historia de usuario Enviar mensaje.

Historias de Usuario	
Número: 9	**Nombre de la HU:** Enviar mensaje
Programador: Miguel Alejandro Diaz Rivera	**Iteración asignada:** 1

Prioridad: alta	**Tiempo estimado:** 9 días
Riesgo en desarrollo: alto	**Tiempo real:** 9 días
Descripción: permite a un usuario autenticado en el sistema, enviar un mensaje de texto a otro usuario registrado en el sistema. De esta manera permite al sistema registrar y almacenar estos mensajes en la base de datos, así como los datos vinculados a los mismos.	
Observaciones: N/A	
Prototipo de interfaz: 	

Fuente: elaboración propia.

2.5. Arquitectura de software

La arquitectura de un software o sistema de cómputo, es la estructura que comprende a los componentes del mismo, sus propiedades externas visibles y las relaciones entre ellos. Es una representación que permite: analizar la efectividad del diseño para cumplir los requerimientos establecidos, considerar alternativas arquitectónicas en una etapa en la que hacer cambios al diseño todavía es relativamente fácil y reducir los riesgos asociados con la construcción del software (Pressman R. , 2010).

La solución propuesta sigue una arquitectura orientada a servicios, específicamente a servicios REST (por sus siglas en inglés*, RepresentationalState Transfer*). Esta arquitectura se compone por dos partes fundamentales, una aplicación del lado del cliente o *frontend*[1], y una aplicación del lado del servidor o *backend*[2]en la forma de una REST-API. El *frontend*es la parte del software que permite al usuario interactuar con la aplicación a través de una interfaz de usuario. Desde el *frontend*el usuario puede, producto de su interacción con el mismo, enviar información a la aplicación servidor, el *backend.* El *backend* a su vez procesa la información recibida desde el *frontend*y devuelve al mismo nueva información para ser procesada en el lado del cliente en función de los requerimientos de la interfaz de usuario.

El intercambio de datos entre el cliente y el servidor se lleva a cabo a través del protocolo de transferencia de datos HTTP (por sus siglas en inglés*, Hypertext Transfer Protocol*), y el formato en el cual tiene lugar este intercambio es JSON (por sus siglas en inglés*JavascriptObjectNotation*). En la figura 1 se observa la arquitectura de la solución propuesta.

[1] Parte de la aplicación del lado cliente.
[2] Parte de la aplicación del lado servidor.

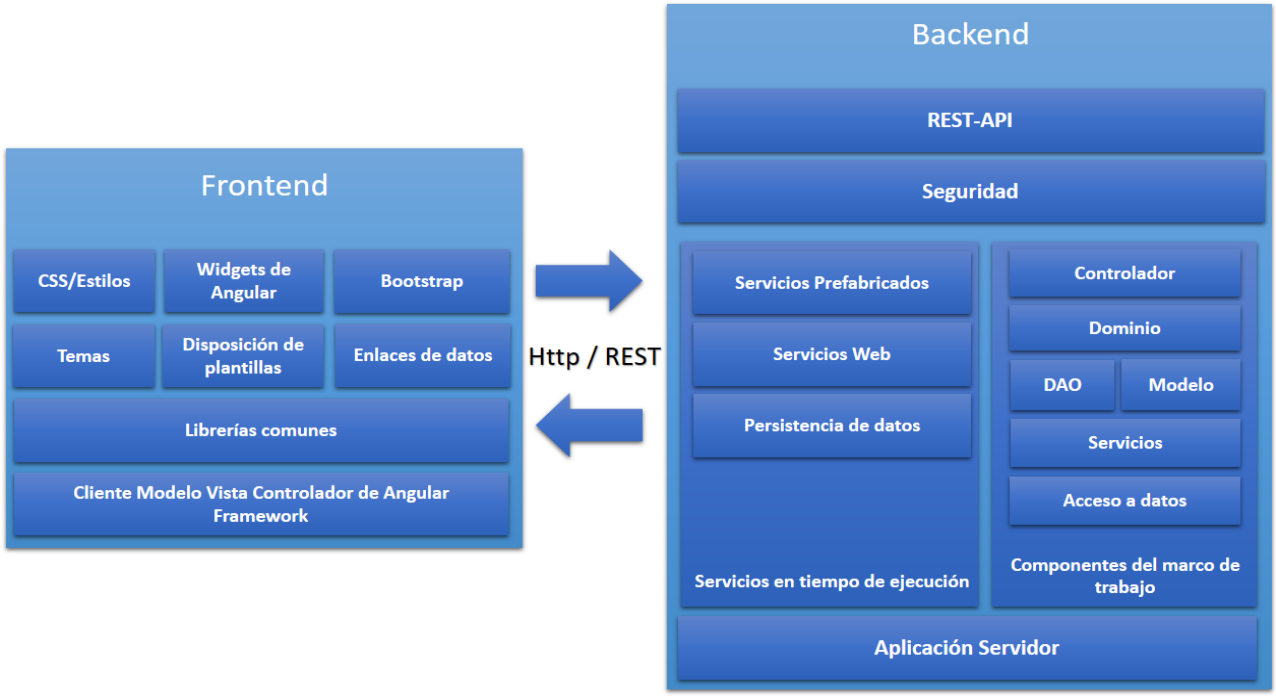


Figura 1. Arquitectura del backend y el frontend de la solución propuesta.
Fuente:elaboración propia.

Angular implementa el patrón Modelo Vista Controlador (*MVC,*por sus siglas en inglés, *Model View Controller*). El patrón Modelo Vista Controlador es uno de los patrones arquitectónicos más usados en el desarrollo de software, tanto en la implementación de diversas aplicaciones como en el diseño de diversos marcos de trabajo. Este patrón consiste en separar la lógica del negocio de la interfaz de usuario, facilitando la evolución por separado de la flexibilidad y la reutilización del código. El flujo de control de este patrón comienza cuando el usuario realiza una acción en la interfaz, produciendo un evento. Luego el controlador trata el evento ocurrido y notifica al modelo la acción del usuario, lo que puede implicar un cambio en el estado del modelo. Seguidamente se genera una vista, que a su vez utiliza los datos del modelo. De esta manera se repite el ciclo cada vez que el usuario realice una acción en la interfaz (UA, 2019)(ASP.NET, 2017).

En la figura 2 se observa en detalle cómo se pone de manifiesto este patrón en la arquitectura del *frontend* de la solución propuesta, tomando como ejemplo el RF “Enviar Mensaje”, teniendo en cuenta que este es uno de los RF que implementa una interfaz de usuario sobre la cual se puede ejemplificar el uso del patrón arquitectónico antes descrito.

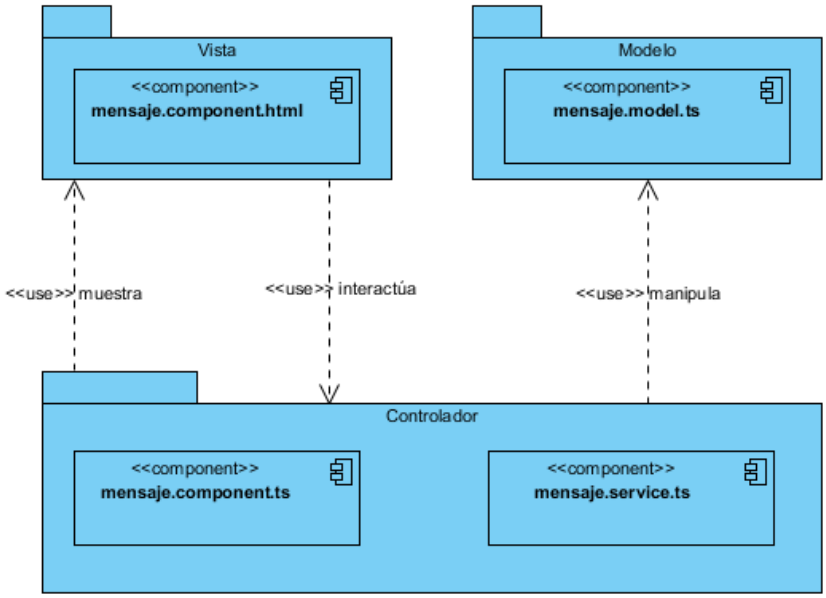


Figura 2. Arquitectura del frontend para el RF_Enviar Mensaje.
Fuente: elaboración propia.

En la figura 2 se observan los tres componentes principales que forman la arquitectura Modelo Vista Controlador del *frontend* de la solución propuesta. El modelo está compuesto por el archivo *mensaje.model.ts,* el cual contiene la información y atributos que necesita el sistema sobre los mensajes, para enviar o recibir los mismos. Por otra parte, la vista se compone por el archivo *mensaje.component.html*, este representa la información que se muestra al usuario en la interfaz e interactúa con funciones específicas del controlador. El controlador está compuesto por el archivo *mensaje.component.ts,* encargado de interactuar con el modelo y mostrar la información a la vista.

De igual manera, *Spring Boot* implementa el patrón N-Capas. Este patrón se enfoca principalmente en el agrupamiento de funcionalidades relacionadas dentro de una aplicación en distintas capas que son colocadas verticalmente una encima de otra. La funcionalidad dentro de cada capa se relaciona con un rol o responsabilidad especifica. En una aplicación con una capa lógica, una capa de servicios y una capa de acceso a datos, la responsabilidad de la capa de la lógica del negocio es la de hacer cumplir las reglas del negocio o requerimientos de la aplicación. La capa de servicio se encarga de implementar la manera en que se realizan operaciones de transacción de datos con la base de datos.

Por otra parte, la responsabilidad de la capa de acceso a datos es la de recuperar y modificar directamente los datos desde su origen. La principal característica de este tipo de arquitectura es que los servicios que ofrece una capa N determinada solo son utilizados por la capa N+1, y

en consecuencia la responsabilidad de cada capa es brindar sus servicios exclusivamente a la capa inmediata superior. De esta manera se eliminan las dependencias directas entre capas(Serafin, 2018).

En la figura 3 se observa en detalle cómo se pone de manifiesto este patrón en la arquitectura del *backend* de la solución propuesta, para el RF "Enviar Mensaje", teniendo en cuenta que este es uno de los requisitos funcionales que implementa una interfaz de usuario sobre la cual se puede ejemplificar el uso del patrón arquitectónico antes descrito.

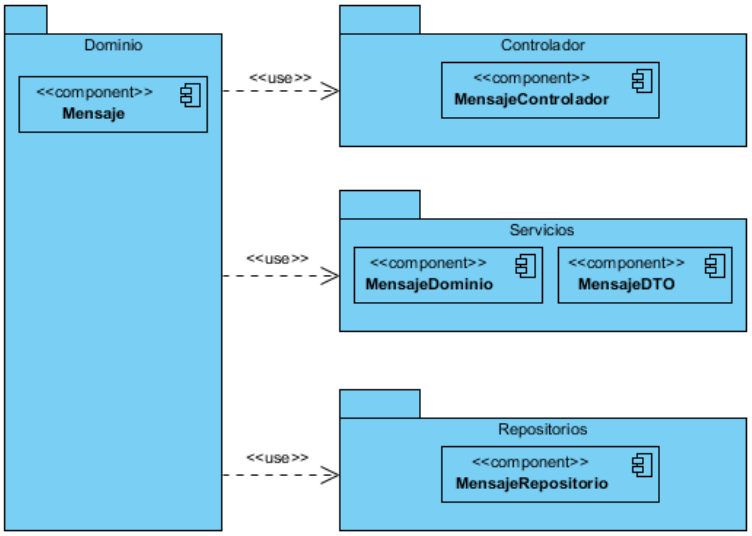


Figura 3. Arquitectura del *backend* para el *RF_Enviar Mensaje.*
Fuente: elaboración propia

En la figura 3 se observa cómo se delegan distintas responsabilidades a cada capa de la arquitectura del *backend* de la solución propuesta. Por una parte, el componente *MensajeControlador*de la capa Controlador contiene la lógica del negocio relacionada con el flujo de información entre la aplicación cliente y el lado del servidor, y utiliza además el componente *MensajeDominio* de la capa Servicios. La capa Servicios contiene a los componentes que definen la lógica del negocio relacionada con la recuperación de datos. El componente *MensajeDominio* contiene la información que necesita el sistema acerca de las entidades necesarias para realizar la transacción de datos entre la aplicación y la base de datos, relacionadas con el envío de mensajes. La capa Repositorios o capa de Acceso a Datos contiene el componente *MensajeRepositorio,* el cual se encarga de manejar las operaciones atómicas de recuperación de datos entre la aplicación y la base de datos. A su vez la capa Dominio o capa de dominio actúa como contenedor de todas aquellos

componentes que representan entidades en el negocio, y que son utilizados a su vez por las capas antes mencionadas.

2.6. Análisis y diseño

En esta disciplina se modela el sistema y su forma (incluida su arquitectura) para que soporte todos los requisitos, incluyendo los requisitos no funcionales(Pressman R. , 2010). Esta etapa de desarrollo del software representa el enlace entre los requisitos y la implementación del sistema teniendo como base los patrones de diseño definidos por parte del equipo de arquitectura del proyecto en el que se enmarca la solución propuesta. A continuación, se describen estos patrones:

2.6.1. Patrones de diseño GRASP

Patrón Controlador:el objetivo de este patrón es asignar la responsabilidad de controlar el flujo de eventos del sistema, a clases específicas. Delega la responsabilidad en otras clases con las que mantiene un modelo de alta cohesión. Sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa, de tal forma que es la que recibe los datos del usuario y la que los envía a las distintas clases según el método llamado(Larman, 2016). En la solución propuesta este patrón es utilizado a través de las clases controladoras ubicadas en la aplicación del lado del servidor, las cuales, a través de métodos permiten a la aplicación del lado del cliente acceder a distintas funcionalidades. En la figura 4se muestra un fragmento de código de la clase *MensajeControlador.java* donde se evidencia el uso de este patrón en la solución propuesta.

```java
@CrossOrigin(origins = "*")
@RestController("/mensaje")
public class MensajeControlador {

    @Autowired
    MensajeService mensajeService;

    @GetMapping("/listar/conversacion/{id}")
    public List<Mensaje> listarMensajesPorConversacion(@PathVariable("id") Long idConversacion) {
        mensajeService.cambiarEstadoMensajesPorConversacion(idConversacion);
        List<Mensaje> mensajes = mensajeService.listarMensajesPorConversacion(idConversacion);
        return mensajes;
    }

    // ... ///
}
```

Figura 4. Aplicación del patrón controlador en la clase *MensajeControlador.java*
Fuente: elaboración propia

Patrón Experto: este patrón define la clase experta en información, dado que esta es la que contiene toda la información para cumplir con la responsabilidad. De este modo se obtendrá un diseño con mayor cohesión y así la información se mantiene encapsulada(Larman, 2016). En la solución propuesta este patrón se evidencia en las entidades de la aplicación, las cuales en el caso de la aplicación *backend*, son clases de extensión *.java*, que contienen la información necesaria para cumplir con su función. En la figura 5 se muestra un fragmento de la clase *Mensaje.java*, que constituye una entidad del sistema.

```java
@Data
@Entity
public class Mensaje {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    private Long idReceptor;
    private Long idCreador;
    @NotEmpty(message = "El contenido del mensaje no debe ser vacio")
    @Size(max = 500, message = "El contenido del mensaje no debe exceder los 500 caracteres")
    private String contenido;
    private Date fechaCreacion;

}
```

Figura 5. Aplicación del patrón experto en la clase *Mensaje.java*
Fuente: elaboración propia

Patrón Creador:permite identificar quién debe ser el responsable de la creación (o instanciación) de nuevos objetos o clases(Larman C. , 2016). En la solución propuesta este patrón se hace evidente en las clases creadoras, sufijadas con la palabra "*Factory*", y cuyos métodos y funcionalidades tienen el propósito de crear una instancia de un objeto determinado a partir de un conjunto de datos o un objeto externo a la clase. En la figura 6se muestra un fragmento de código de la clase *MensajeFactory.java*, donde se evidencia el uso de este patrón en el *backend*.

```java
@Component
public class MensajeFactory {
    public Mensaje createFromDTO(MensajeDTO mensajeDTO) {
        return Mensaje.builder()
                .idCreador(mensajeDTO.getIdCreador())
                .idReceptor(mensajeDTO.getIdReceptor())
                .nombreCreador(mensajeDTO.getNombreCreador())
                .nombreReceptor(mensajeDTO.getNombreReceptor())
                .contenido(mensajeDTO.getContenido())
                .leido(false)
                .build()
                ;
    }
}
```

Figura 6. Aplicación del patrón creador en la clase *MensajeFactory.java*

Fuente: elaboración propia.

Patrón bajo acoplamiento: este patrón plantea la baja dependencia que debe existir entre las clases y está estrechamente relacionado con los patrones Experto o Alta Cohesión(Larman C. , 2016).

En la presente investigación el uso de este patrón se evidencia en el 30 % de las clases del diseño, las cuales tienen una baja dependencia una de otras, tal y como se demuestra en el Capítulo 3 con los resultados de la validación del diseño a partir de la aplicación de métricas.

Patrón alta cohesión: este patrón permite asignar responsabilidades de manera que la cohesión siga siendo alta (Larman C. , 2016)

En la presente investigación el uso de este patrón se evidencia en el 30 % de las clases del diseño, las cuales tienen una baja sobrecarga de responsabilidades, tal y como se demuestra en el Capítulo 3 con los resultados de la validación del diseño a partir de la aplicación de métricas.

2.6.2. Patrones de diseño GoF

Patrón decorador:permite añadir responsabilidades a objetos concretos de manera dinámica y transparente sin afectar a otros objetos. Este patrón brinda más flexibilidad que la herencia estática y evita que las clases más altas en la jerarquía estén demasiado cargadas de funcionalidad y sean complejas(Larman, 2016). En el caso de la presente investigación el uso este patrón se evidencia mayormente en la utilización de anotaciones provistas por el marco de trabajo *Spring Boot*. En la figura 7se muestra un fragmento de código donde se evidencia este patrón en el *backend*. En el ejemplo se pone de manifiesto como a través de la anotación *@Service* se logra que la clase *MensajeServicioImpl.java* sea reconocida por el contexto de la aplicación como un componente único.

```
@Service
public class MensajeServicioImpl implements MensajeService {

    @Autowired
    MensajeRepository mensajeRepository;
    @Autowired
    MensajeService mensajeService;
    @Autowired
    ConversacionRepository conversacionRepository;
    @Autowired
    ConversacionService conversacionService;
    @Autowired
    MensajeFactory mensajeFactory;
```

Figura 7. Aplicación del patrón decorador a través del uso de la anotación *@Service.*

Fuente: elaboración propia

Patrón observador:permite observar los cambios producidos por un objeto, de esta forma, cada cambio que afecte el estado del objeto observado lanzará una notificación a los observadores; a esto se le conoce como Publicador-Suscriptor(Larman C. , 2016). En la solución propuesta este patrón se evidencia en el uso de la clase *SystemEventListener.java*, a través de la cual se detecta el acceso de determinados objetos al contexto de la aplicación y se publica un objeto en respuesta a dicho evento. En la figura 8 se muestra un fragmento de código donde se evidencia el uso de este patrón.

```
@Service
public class SystemEventListener implements ApplicationEventPublisherAware {

    @Autowired
    private ApplicationEventPublisher publisher;
    @Autowired
    private SystemEventAdapter systemEventAdapter;

    @EventListener
    public void listen(SystemEvent systemEvent) {
        NotificacionDTO notificacionDTO = systemEventAdapter.convert(systemEvent);
        publisher.publishEvent(notificacionDTO);
    }
```

Figura 8. Aplicación del patrón observador en la clase *SystemEventListener.java.*
Fuente: elaboración propia

2.6.3. Otros patrones

Patrón inyección de dependencias:es un patrón de diseño usado en la programación orientada a objetos, que trata de solucionar las necesidades de creación de los objetos de una manera práctica, útil, escalable y con una alta versatilidad del código. Mediante el uso de este patrón se puede, desde cualquier parte de la aplicación, utilizar instancias de objetos sin la necesidad de instanciarlos directamente en dicho lugar(Larman C. , 2016). Tanto Angular como *Spring Boot* tienen su propia implementación de la inyección de dependencias, haciendo uso de técnicas propias acordes al contexto y lenguaje de cada una de estas tecnologías.En Angular esto significa que se pueden requerir servicios u objetos que alguna de las clases necesita, ya sean componentes, directivas o servicios, sin la necesidad de instanciar estas dependencias. En *Spring Boot*se puede observar como a través de la anotación *@Autowired* sobre la declaración de una variable, se obtiene acceso de esta variable como una instancia de la clase de la que hereda, sin necesidad de instanciarla manualmente. En la figura 9se muestra un fragmento de código donde se expone, a través de la clase *WebSocketMensajeControlador.java*, el uso de este patrón en la solución propuesta.

```java
@Controller
public class WebSocketMensajeControlador {

    @Autowired
    MensajeService mensajeService;
    @Autowired
    ConversacionService conversacionService;
    @Autowired
    ConversacionRepository conversacionRepository;
```

Figura 9. Aplicación del patrón inyección de dependencias en la clase *WebSocketMensajeControlador.java.*
Fuente: elaboración propia

2.7. Diagrama de clases del diseño

El diagrama de clases del diseño describe gráficamente las especificaciones de las clases de software y las interfaces que participan en el sistema con sus relaciones estructurales y de herencia. Los diagramas de este tipo contienen las definiciones de las entidades del software en vez de conceptos del mundo real y son utilizados durante el proceso de análisis y diseño de los sistemas, donde se crea una vista lógica de la información que se maneja en el sistema, los componentes que se encargarán del funcionamiento y la relación entre uno y otro (Larman C. , 2016).

En el diagrama de clases de la figura 10 se muestra la estructura de las clases que intervienen en la HU Enviar mensaje. En la figura se pueden observar las distintas clases, componentes y su interacción entre si cuando se envía un mensaje en el sistema. En la vista se observa como el archivo *mensaje.service.ts* es el encargado de establecer una conexión, y/o hacer una petición a la clase controladora en el *backend, WebSocketMensajeControlador.java*, a través del protocolo *websocket*[3]y utilizando el formato de intercambio de texto JSON. Una vez arribada la petición al *backend*, la clase controladora se nutre tanto del modelo como los servicios y repositorios representados en la figura para ejecutar la funcionalidad deseada.

[3] Tecnología que proporciona un canal de comunicación bidireccional.

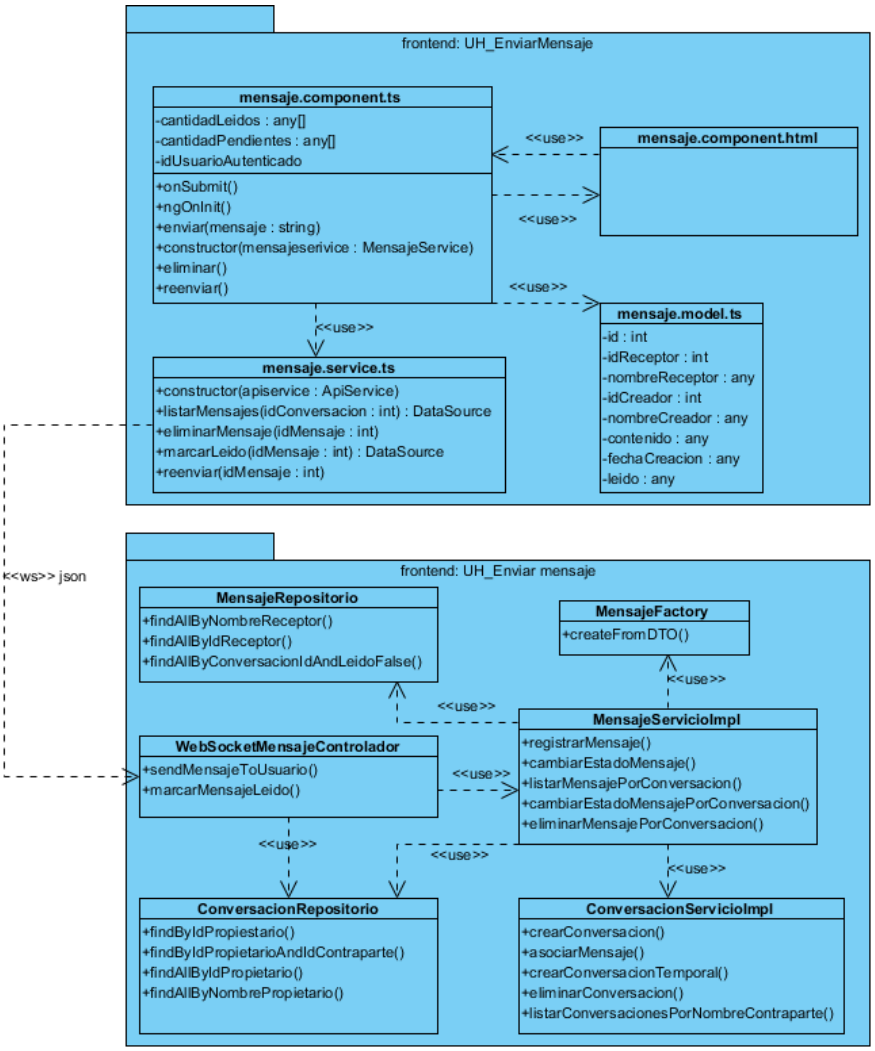


Figura 10. Diagrama de clases del diseño para la *HU_Enviar mensaje.*
Fuente: elaboración propia.

2.8. Modelo de Datos

El modelo de datos se utiliza para describir la estructura lógica y física de la información persistente gestionada por el sistema. Se utiliza para definir la correlación entre las clases de diseño persistentes y las estructuras de datos persistentes, y para definir las estructuras de datos persistentes (Lucid_Software_Inc, 2019).

En la figura 11se muestra el modelo de datos que define la estructura de la información en la solución propuesta

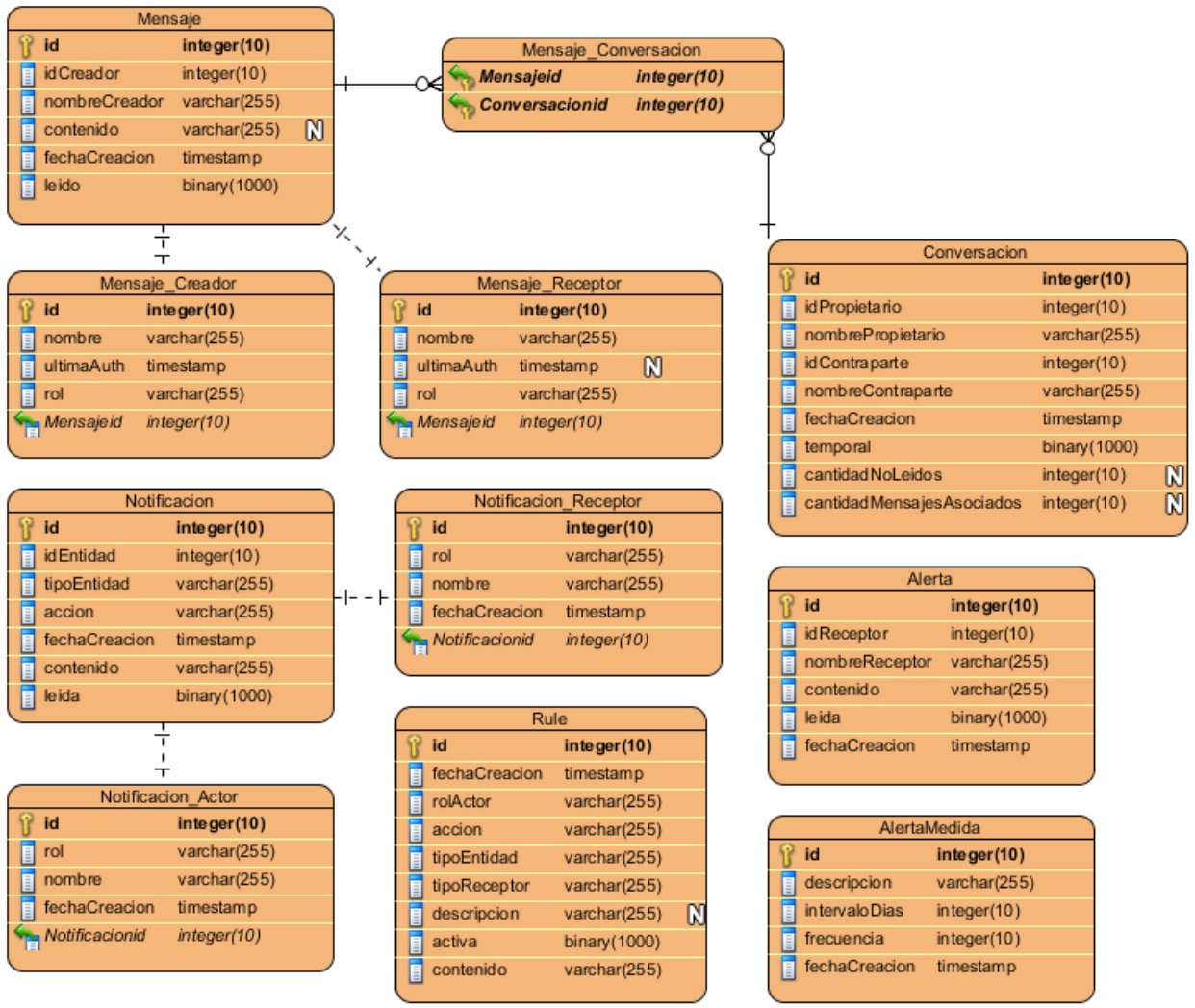


Figura 11. Modelo de datos de la solución propuesta.
Fuente: elaboración propia.

El modelo de datos obtenido contiene un total de 11 tablas. Las tablas *Notificación* y *Alerta* almacenan los datos relacionados a las notificaciones y alertas respectivamente, generadas por el sistema. La tabla *AlertaMedida* contiene los datos necesarios para personalizar la funcionalidad de generar una alerta, acorde a las prioridades e intereses establecidos. Las tablas *Notificacion_Actor* y*Notificacion_Receptor* contienen datos necesarios para el proceso acerca de los usuarios que desempeñan los roles de actor y receptor de la notificación generada. Por otra parte, la tabla *Rule* contiene los datos contra los cuales se comprobarán los datos de un evento ocurrido para determinar si, y como se debe generar una notificación. La tabla *Mensaje* contiene los datos relativos a los mensajes enviados de un usuario a otros, mientras que las tablas*Mensaje*_Creador y Mensaje_*Receptor* contiene datos relativos a estos usuarios. Por último, la tabla *Conversacion* contiene los datos necesarios para agrupar y gestionar los mensajes que se envían entre dos usuarios en particular.

2.9. Estándares de codificación

Los estándares de codificación constituyen buenas prácticas o conjunto de reglas no formales que han ido surgiendo en las comunidades de desarrolladores de software con el paso del tiempo. Estos tienen el propósito de obtener un código fuente más legible, portable, seguro, eficiente, robusto y cohesionado, permitiendo mayor velocidad en el desarrollo, mejor coordinación entre los equipos de trabajo.Además,logran que para un desarrollador sea más fácil integrarse a un proyecto ya comenzado, se eviten bugs[4] y se faciliten los procesos de mantenibilidad y revisión de código(Hommel S., 2019). Este grupo de buenas prácticas pueden ser definidas en dependencia del lenguaje o marco de trabajo a utilizar.

Los estándares de codificación utilizados en el desarrollo del componente propuesto fueron definidos por el equipo de desarrollo del SIGAP. A continuación, se describen los mismos:

- Los nombres de las clases comenzarán con mayúscula, en caso de ser un nombre compuesto las siguientes palabras se escribirán de igual forma. En la figura 12 se evidencia el uso de este estándar en la clase *MensajeRepository.java*.

[4]Error de software que desencadena un resultado indeseado.

```
@Repository
public interface MensajeRepository extends JpaRepository<Mensaje, Long> {
```

Figura 12. Aplicación de estándar de codificación en el nombre de la clase *MensajeRepository.java*
Fuente: elaboración propia.

- Los nombres de los métodos comenzarán con minúscula, en caso de ser un nombre compuesto las siguientes palabras se escriben con mayúscula. En la figura 13 se evidencia el uso de este estándar en la clase *MensajeRepository.java*.

```
@Repository
public interface MensajeRepository extends JpaRepository<Mensaje, Long> {

    public Mensaje findByNombreReceptor(String nombreReceptor);
```

Figura 13. Aplicación de estándar de codificación en el método *findByReceptor.*
Fuente: elaboración propia.

- Los identificadores para las variables y los parámetros se escribirán con letras en minúsculas y en caso de ser un nombre compuesto las siguientes palabras se escribirán de igual forma, en la figura14se evidencia el uso de este estándar de codificación.

```
public ResponseEntity<?> registrarMensaje(MensajeDTO mensajedto) {
        try {
            Mensaje mensaje = mensajeRepository.save(mensajeFactory.createFromDTO(mensajedto));
            if (mensaje == null)
                return new ResponseEntity<Mensaje>(mensaje, HttpStatus.BAD_REQUEST);
            else {
                Long idcreador = mensaje.getIdCreador();
                Long idreceptor = mensaje.getIdReceptor();
                String nombrecreador = mensaje.getNombreCreador();
                String nombrereceptor = mensaje.getNombreReceptor();
```

Figura 14. Aplicación de estándar de codificación en identificadores y parámetros.
Fuente: elaboración propia.

- Tanto los nombres de los métodos como de las variables deben ser suficientemente descriptivos en cuanto a su función, y no deben exceder los 15 caracteres.
- Todas las funciones deben tener comentarios, que describan su propósito. En la figura15se evidencia el uso de este estándar en el método*cambiarEstadoMensaje ()* de la clase *MensajeServiceImpl.java*.

```java
/**
 *Asignar al atributo "leido" de un mensaje el valor "true"
 * @param mensaje
 * @return Mensaje
 */
public Mensaje cambiarEstadoMensaje(Mensaje mensaje) {
    mensaje.setLeido();
    return mensajeRepository.save(mensaje);
}
```

Figura 15. Aplicación de estándar de codificación en el método *cambiarEstadoMensaje ()* de la clase *MensajeServiceImpl.java.*
Fuente: elaboración propia.

2.10. Conclusiones parciales

Con el uso de la metodología AUP para la UCI a través del escenario número 4 se logró describir el proceso de desarrollo de los componentes propuestos, lo cual permitió una mayor organización en la implementación de la solución. Por otra parte, la especificación de los requisitos, así como la descripción de los componentes para la gestión de alertas, mensajes y notificaciones propiciaron un mayor entendimiento del proceso a informatizar. La descripción de la arquitectura y sus componentes, permitieron una visión más clara de la implementación deseada y garantizaron la obtención de una solución con poca dependencia entre clases, flexible al mantenimiento y a la introducción de cambios.

CAPÍTULO 3: VALIDACIÓN DE LA SOLUCIÓN PROPUESTA

3.1. Introducción

En el presente capítulo se muestran los resultados obtenidos luego de aplicar las métricas para la validación del diseño de la solución propuesta, documentándose los resultados obtenidosen cada caso. Por otra parte, se aplican las pruebas de software organizadas a partir de las disciplinas establecidas por la metodología *AUP* variación UCI para esta etapa, en este caso solo se realizan Pruebas internas en el nivel de unidad. El capítulo concluye con la verificación del cumplimiento del objetivo general de la investigación a partir de la definición de un conjunto de indicadores que permiten evaluar a través de un antes y un después al desarrollo de los componentes, la relación causa-efecto de la variable independiente sobre las variables dependientes de la investigación.

3.2. Validación del diseño

La validación del diseño a través de las métricas, permite medir de forma cuantitativa la calidad de los atributos internos del software, de forma tal que pueden ayudar al desarrollador a juzgar la calidad de un diseño a nivel de componente (Pressman R. , 2010). Para el caso de la presente investigación se aplicaron las métricas Tamaño Operacional de Clase (TOC) y Relaciones entre Clases (RC) debido al conjunto de atributos de calidad de diseño que ambas miden.A continuación, se describe cómo se llevan a cabo en la solución propuesta.

Relaciones entre clases (RC)

La métrica RC está dada por el número de relaciones de uso de una clase con otra. El primer paso es evaluar un conjunto de atributos de calidad entre los que se encuentran el acoplamiento, complejidad de mantenimiento y reutilización de cada clase(Pressman R. , 2010).

A continuación, se exponen los pasos que se llevaron a cabo para aplicar la métrica a todas las clases de los componentes propuestos en la presente investigación:

- Determinar la Cantidad de Relaciones de Uso (CRU) que poseen las clases a medir.
- Calcular el promedio de las CRU.

- Teniendo en cuenta los valores antes obtenidos se determina la incidencia de los atributos de calidad en cada una de las clases, según los criterios expuestos en la tabla 5.

En la tabla 4 se muestra tanto el total de clases como el total de asociaciones de uso.

Tabla 4. Total de clases y promedio de asociaciones de uso con la aplicación de la métrica RC.

Total de clases	50
Total de asociaciones de uso	84
Promedio de asociaciones de uso	1.68

Fuente: elaboración propia

Tabla 5. Rango de valores para medir la afectación de los atributos de calidad (RC).

Atributos de calidad	Clasificación	Criterio
Acoplamiento	Ninguna	CRU = 0
	Baja	CRU = 1
	Media	CRU = 2
	Alta	CRU > 2
Complejidad de mantenimiento	Baja	CRU <= Promedio
	Media	Promedio < CRU < = 2* promedio
	Alta	CRU > 2* promedio
Reutilización	Baja	CRU > 2* promedio
	Media	Promedio < CRU < = 2* promedio
	Alta	CRU <= Promedio

Cantidad de pruebas	Baja	CRU <= Promedio
	Media	Promedio < CRU < = 2* promedio
	Alta	CRU > 2* promedio

Fuente: (Lorenz, et. al, 1994)

La Tabla 6muestra las medidas de los parámetros de calidad obtenidas luego de aplicar la métrica RC al diseño de clases obtenido en la solución propuesta, en la cual se representan las clases que forman parte de este diseño. En la misma se utilizaron las abreviaturas Ac (Acoplamiento), CM (Complejidad de mantenimiento) y R (Reutilización) y CP (Cantidad de pruebas).

Tabla 6. Resultados obtenidos luego de aplicada la métrica RC.

No.	Clase	CantidaddeRelacionesdeUso	Ac	Cm	R	Cp
1	*enviar-mensaje.component.html*	2	Media	Media	Media	Media
2	*enviar-mensaje.component.ts*	3	Alta	Media	Media	Media
3	*listar-mensaje.component.html*	1	Baja	Baja	Alta	Baja
4	*listar-mensaje.component.ts*	1	Baja	Baja	Alta	Baja
5	*reenviar-mensaje.component.html*	4	Alta	Alta	Baja	Alta
6	*reenviar-mensaje.component.ts*	2	Media	Media	Media	Media
7	*mensaje.ts*	0	Ninguna	Baja	Alta	Baja
8	*mensaje.service.ts*	4	Alta	Alta	Baja	Alta
9	*listar-notificacion.component.html*	2	Media	Media	Media	Media
10	*listar-notificacion.component.ts*	3	Alta	Media	Media	Media
11	*notificacion.ts*	0	Ninguna	Baja	Alta	Baja
12	*notificacion.service.ts*	5	Alta	Alta	Baja	Alta
13	*listar-alerta.component.html*	2	Media	Media	Media	Media
14	*listar-alerta.component.ts*	3	Alta	Media	Media	Media

15	*alerta.ts*	0	Ninguna	Baja	Alta	Baja
16	*alerta.service.ts*	5	Alta	Alta	Baja	Alta
17	*listar-conversacion.component.html*	2	Media	Media	Media	Media
18	*listar-conversacion.component.ts*	2	Media	Media	Media	Media
19	*conversacion.ts*	0	Ninguna	Baja	Alta	Baja
20	*conversacion.service.ts*	3	Alta	Media	Media	Media
21	*WebSocketMensajeControlador.java*	2	Media	Media	Media	Media
22	*WebSocketStompConfiguration.java*	0	Ninguna	Baja	Alta	Baja
23	*MensajeControlador.java*	4	Alta	Alta	Baja	Alta
24	*Mensaje.java*	1	Baja	Baja	Alta	Baja
25	*MensajeFactory.java*	2	Media	Media	Media	Media
26	*MensajeServicio.java*	1	Baja	Baja	Alta	Baja
27	*MensajeServicioImpl.java*	5	Alta	Alta	Baja	Alta
28	*MensajeRepositorio.java*	1	Baja	Baja	Alta	Baja
29	*MensajeDTO.java*	1	Baja	Baja	Alta	Baja
30	*Alerta.java*	0	Ninguna	Baja	Alta	Baja
31	*AlertaFactory.java*	2	Media	Media	Media	Media
32	*AlertaControlador.java*	3	Alta	Media	Media	Media
33	*AlertaServicio.java*	1	Baja	Baja	Alta	Baja
34	*AlertaServicioImpl.java*	3	Alta	Media	Media	Media
35	*AlertaRepositorio.java*	1	Baja	Baja	Alta	Baja
36	*AlertaMedida.java*	1	Baja	Baja	Alta	Baja
37	*AlertaMedidaRepositorio.java*	1	Baja	Baja	Alta	Baja
38	*Conversacion.java*	1	Baja	Baja	Alta	Baja
39	*ConversacionControlador.java*	3	Alta	Media	Media	Media

40	*ConversacionServicio.java*	1	Baja	Baja	Alta	Baja
41	*ConversacionServicioImpl.java*	4	Alta	Alta	Baja	Alta
42	*ConversacionRepositorio.java*	1	Baja	Baja	Alta	Baja
43	*ConversacionDTO.java*	1	Baja	Baja	Alta	Baja
44	*Notificacion.java*	0	Ninguna	Baja	Alta	Baja
45	*NotificacionServicio.java*	0	Ninguna	Baja	Alta	Baja
46	*NotificacionServicioImpl.java*	2	Media	Media	Media	Media
47	*NotificacionRepositorio.java*	0	Ninguna	Baja	Alta	Baja
48	*NotificacionDTO.java*	0	Ninguna	Baja	Alta	Baja
49	*NotificacionGenerador.java*	4	Alta	Alta	Baja	Alta
50	*SistemaNotificacionUtils.java*	1	Baja	Baja	Alta	Baja

Fuente: elaboración propia.

En la figura 16 se muestra a través de un gráfico de barras los resultados de la aplicación de la métricaRelación de Clases.

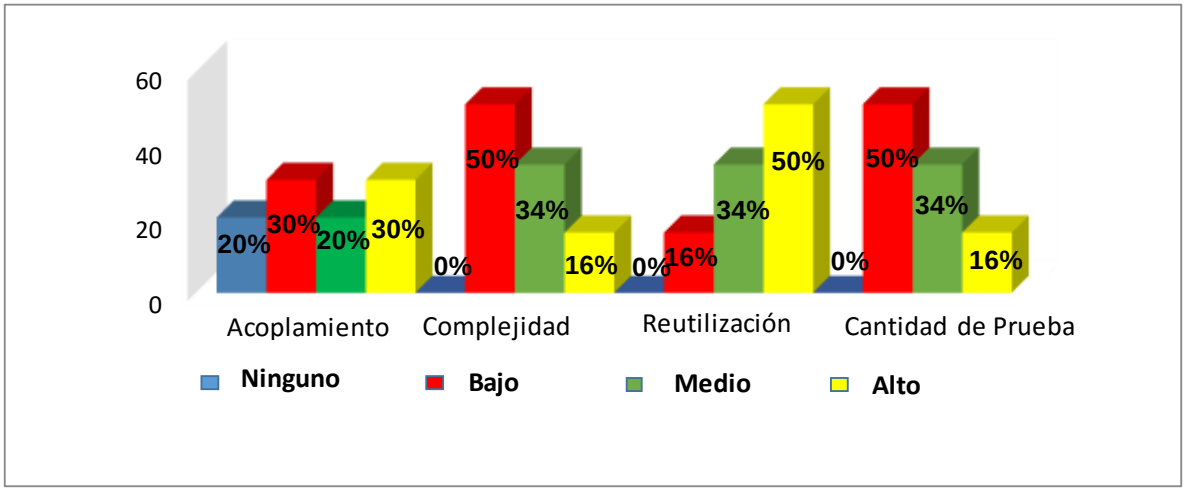


Figura 16. Representación en (%) de los resultados de la aplicación de la métrica RC.
Fuente: elaboración propia.

Acoplamiento: según los resultados que se muestran, el 20% de las clases no posee relaciones de uso y el 30 % posee un acoplamiento bajo, el 20% un acoplamiento medio y el 30% un acoplamiento alto, por lo que la mayoría de las clases poseen valores de acoplamiento medio y alto, validandouna realización correcta del diseño.

Complejidad de mantenimiento: según los resultados que se muestran en la figura anterior, el 50% de las clases presentan una complejidad de mantenimiento baja, el 34% una complejidad media,y un 16% una complejidad alta, demostrando queson de fácil soporte.

Reutilización: según los resultados que se muestran en la figura, el 16% de las clases tienen un grado de reutilización bajo, mientras que el 34% de las clases tiene un grado de reutilización medio y el 50% un grado alto del mismo atributo.

Cantidad de pruebas: luego de aplicar la métrica se obtuvo que el 50%, el 34% y el 16% de las clases poseen un grado bajo, medio y alto de esfuerzo respectivamente, a la hora de realizar cambios, rectificaciones y pruebas de software.

Teniendo en cuenta lo anterior se concluye que el diseño delos componentes propuestos en la presente investigación alcanzó resultados positivos durante la evaluación de la métrica. Estos datos muestran que las clases poseen bajo acoplamiento,complejidad de mantenimiento, esfuerzo para realizar pruebas y en efecto presentan un alto grado de reutilización.

Tamaño Operacional de Clases (TOC)

La métrica TOC se aplica a cada una de las clases del diseño con el objetivo de medir la calidad de las mismas con respecto a su grado de responsabilidad, complejidad de implementación y reutilización(Pressman R. S., 2002).

A continuación, se explican los pasos que se llevaron a cabo para aplicar la métrica:

- Cálculo del umbral. El umbral se toma del tamaño general de una clase que se determina sumando todas las operaciones que posee.
- Calcular el promedio de los umbrales.
- Teniendo en cuenta los valores antes obtenidos se determina la incidencia de los atributos de calidad en cada una de las clases, según los criterios expuestos en la tabla 8.

En la tabla 7 se muestra tanto el total de clases como el promedio de procedimientos.

Tabla 7. Total de clases y promedio de procedimientos con la aplicación de la métrica TOC.

Total de clases	50
Total de procedimientos	162

Promedio de procedimientos	3.24

Fuente: elaboración propia

Tabla 8. Rango de valores para medir la afectación de los atributos de calidad (TOC).

Atributos de calidad	Clasificación	Criterio
Responsabilidad	Baja	Umbral<= Promedio
	Media	Promedio < Umbral < = 2* promedio
	Alta	Umbral > 2* promedio
Complejidad de implementación	Baja	Umbral <= Promedio
	Media	Promedio < Umbral < = 2* promedio
	Alta	Umbral > 2* promedio
Reutilización	Baja	Umbral > 2* promedio
	Media	Promedio < Umbral < = 2* promedio
	Alta	Umbral <= Promedio

Fuente:(Lorenz & Kidd, 1994)

La Tabla 9 muestra las medidas de los parámetros de calidad obtenidas luego de aplicar la métrica TOC al diseño de clases del requisito que se toma como ejemplo en la presente investigación. En la misma se utilizaron las abreviaturasRp (Responsabilidad), CI (Complejidad de implementación) y R (Reutilización).

Tabla 9. Resultados obtenidos luego de aplicada la métrica TOC.

No.	Clase	Cantidadde Procedimientos	Rp	C	R
1	*enviar-mensaje.component.html*	3	Baja	Baja	Alta

2	*enviar-mensaje.component.ts*	5	Media	Media	Media
3	*listar-mensaje.component.html*	2	Baja	Baja	Alta
4	*listar-mensaje.component.ts*	4	Media	Media	Media
5	*reenviar-mensaje.component.html*	4	Media	Media	Media
6	*reenviar-mensaje.component.ts*	6	Media	Media	Media
7	*mensaje.ts*	1	Baja	Baja	Alta
8	*mensaje.service.ts*	9	Alta	Alta	Baja
9	*listar-notificacion.component.html*	2	Baja	Baja	Alta
10	*listar-notificacion.component.ts*	4	Media	Media	Media
11	*notificacion.ts*	1	Baja	Baja	Alta
12	*notificacion.service.ts*	5	Media	Media	Media
13	*listar-alerta.component.html*	2	Baja	Baja	Alta
14	*listar-alerta.component.ts*	4	Media	Media	Media
15	*alerta.ts*	1	Baja	Baja	Alta
16	*alerta.service.ts*	5	Media	Media	Media
17	*listar-conversacion.component.html*	3	Baja	Baja	Alta
18	*listar-conversacion.component.ts*	4	Media	Media	Media
19	*conversacion.ts*	1	Baja	Baja	Alta
20	*conversacion.service.ts*	7	Alta	Alta	Baja
21	*WebSocketMensajeControlador.java*	2	Baja	Baja	Alta
22	*WebSocketStompConfiguration.java*	2	Baja	Baja	Alta
23	*MensajeControlador.java*	3	Baja	Baja	Alta
24	*Mensaje.java*	2	Baja	Baja	Alta
25	*MensajeFactory.java*	1	Baja	Baja	Alta
26	*MensajeServicio.java*	6	Media	Media	Media

27	*MensajeServicioImpl.java*	6	Media	Media	Media
28	*MensajeRepositorio.java*	3	Baja	Baja	Alta
29	*MensajeDTO.java*	1	Baja	Baja	Alta
30	*Alerta.java*	2	Baja	Baja	Alta
31	*AlertaFactory.java*	1	Baja	Baja	Alta
32	*AlertaControlador.java*	2	Baja	Baja	Alta
33	*AlertaServicio.java*	5	Media	Media	Media
34	*AlertaServicioImpl.java*	5	Media	Media	Media
35	*AlertaRepositorio.java*	3	Baja	Baja	Alta
36	*AlertaMedida.java*	2	Baja	Baja	Alta
37	*AlertaMedidaRepositorio.java*	1	Baja	Baja	Alta
38	*Conversacion.java*	2	Baja	Baja	Alta
39	*ConversacionControlador.java*	4	Media	Media	Media
40	*ConversacionServicio.java*	6	Media	Media	Media
41	*ConversacionServicioImpl.java*	6	Media	Media	Media
42	*ConversacionRepositorio.java*	4	Media	Media	Media
43	*ConversacionDTO.java*	1	Baja	Baja	Alta
44	*Notificacion.java*	2	Baja	Baja	Alta
45	*NotificacionServicio.java*	5	Media	Media	Media
46	*NotificacionServicioImpl.java*	5	Media	Media	Media
47	*NotificacionRepositorio.java*	2	Baja	Baja	Alta
48	*NotificacionDTO.java*	1	Baja	Baja	Alta
49	*NotificacionGenerador.java*	3	Baja	Baja	Alta
50	*SistemaNotificacionUtils.java*	1	Baja	Baja	Alta

Fuente: elaboración propia

En la figura 17 se muestraa través de un gráfico de barras los resultados de la aplicación de la métricaTamaño Operacional de Clases.

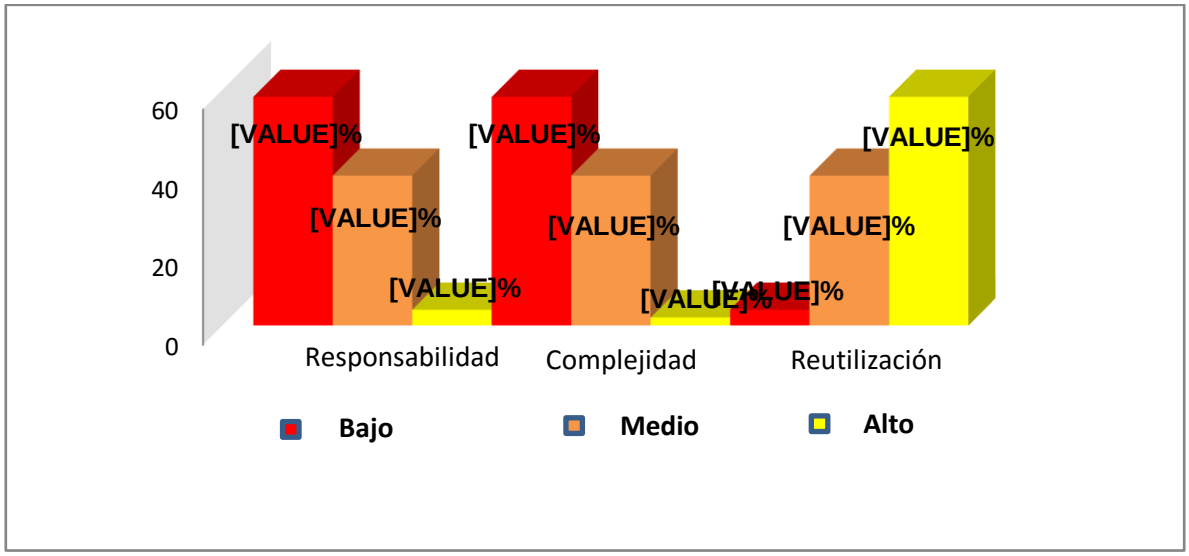


Figura 17. Representación en (%) de los resultados de la aplicación de la métrica TOC.
Fuente: elaboración propia.

Responsabilidad: después de aplicar la métrica, se obtuvieron resultados satisfactorios que reflejan una baja responsabilidad con un valor del 58%, una responsabilidad media de un 38% y una alta responsabilidad de un 4%.

Complejidad de implementación: luego de haber realizado la medición de la métrica, se obtuvieron resultados positivos ya que la complejidad de las clases es baja en un 58%. Siendo media en un 38% y alta en un 2%.

Reutilización:el análisis anterior demuestra que las clases son altamente reutilizables, alcanzando un índice de reusabilidad alta de un 58%, un índice de reusabilidad media de un 38%, y un índice de reusabilidad baja del 4%.

3.3 Pruebas de Software

Para la validación de la solución propuesta se aplican pruebas de software organizadas a partir de las disciplinas definidas para la etapa de pruebas en la metodología AUP variación UCI. Teniendo en cuenta que la solución propuesta solo llegó hasta la implementación del *backend*, la misma fue validada solamente a través de Pruebas internas a nivel de unidad aplicando el método de Caja blanca.

3.3.1 Pruebas a nivel de unidad

La prueba de unidad se concentra en el esfuerzo de verificación de la unidad más pequeña del diseño: el componente o módulo de software. Tomando como guía la descripción del diseño a nivel de componente, se prueban importantes caminos de control para describir errores dentro de los límites del módulo. El alcance restringido que se ha determinado para las pruebas de unidad limita la relativa complejidad de las pruebas y los errores que éstas descubren (Pressman R. , 2010).

Método de caja blanca

El método de caja blanca, en ocasiones llamada pruebas de cristal, es un método que usa la estructura de control descrita como parte del diseño al nivel de componentes para derivar los casos de prueba(Pressman R. , 2010).

La técnica de ruta básica es empleada en el método de Caja blanca, la misma tiene como objetivo comprobar que cada camino se ejecute independiente de un componente o programa, obteniéndose una medida de la complejidad lógica del diseño. Esta técnica de prueba debe ser utilizada para evaluar la efectividad de los métodos asociados a una clase, con el objetivo de asegurar que cada camino independiente sea ejecutado por lo menos una vez en el sistema (Pressman R. , 2010). Al emplear los métodos de esta prueba se derivan casos de pruebas que:

- Garantizan que todas las rutas independientes dentro del módulo se han ejercitado por lo menos una vez.
- Ejercitan los lados verdaderos y falsos de todas las decisiones lógicas.
- Ejecutan todos los bucles en sus límites y dentro de sus límites operacionales.
- Ejercitan estructuras de datos internos para asegurar su validez(Pressman R. , 2010).

En la validación de los componentes propuestos, la técnica de camino básico se aplicó a todos los métodos de las clases controladoras, teniendo en cuenta que estos agrupan las principales funcionalidades de la solución. A continuación, se muestran los resultados de la aplicación de la técnica sobre la funcionalidad *registrarMensaje*perteneciente a la clase *MensajeServiceImpl.java*. Se toma este método como ejemplo teniendo en cuenta que responde a una de las principales funcionalidades del componente.

La complejidad ciclomática obtenida con la aplicación de la técnica indica la cantidad de caminos independientes a recorrer para probar el código analizado. Es decir, se obtiene el número de casos de pruebas a desarrollar para la validación del método. En lafigura 18 se muestra el código del método *registrarMensaje*de la clase *MensajeServiceImp.javal*.

Figura 18. Método registrarMensaje de la clase *MensajeServiceImp.javal.*
Fuente: elaboración propia.

A continuación, se detallan los pasos que se utilizaron al aplicarla técnica ruta básica:

1. **Confeccionar el grafo de flujo**: usando el código de la Figura 18 se realizó la representación del grafo de flujo, el cual describe un flujo de control lógico y está compuesto por los siguientes elementos:
   - Nodos de gráfica de flujo: son círculos que representan una o más instrucciones procedimentales.
   - Aristas o enlaces: son flechas que representan el flujo de control y sonanálogas a las flechas del diagrama de flujo.
   - Regiones: son las áreas delimitadas por aristas y nodos.

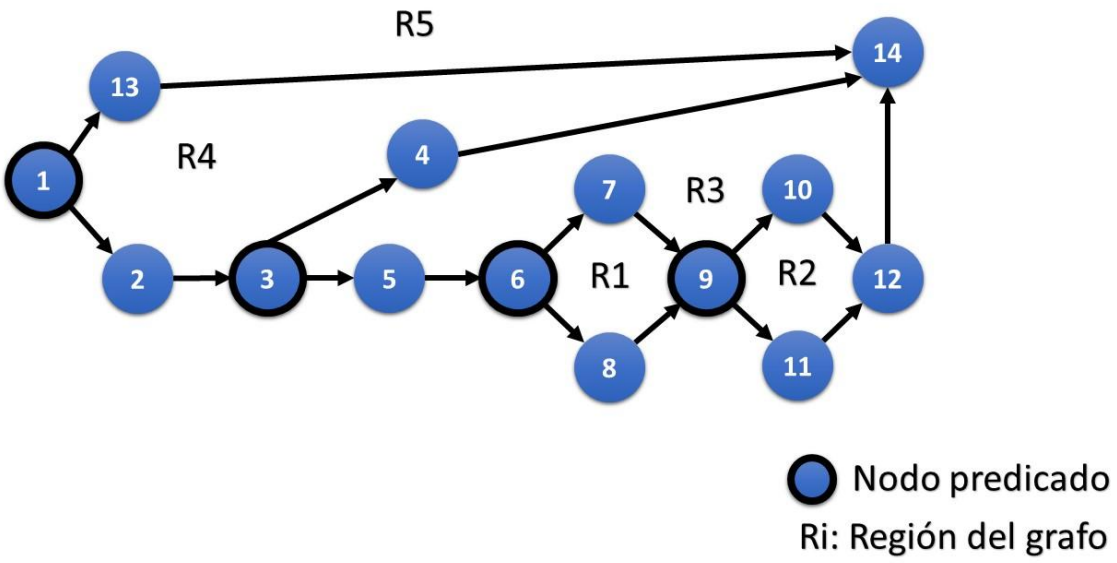


Figura 19. Grafo de la ruta básica del método *registrarMensaje.*
Fuente: elaboración propia.

2. **Calcular la complejidad ciclomática**: proporciona una medición cuantitativa de la complejidad lógica de un programa. El valor calculado define el número de caminos independientes del conjunto básico de un programa, y proporciona un límite superior para el número de pruebas que deben aplicarse para asegurar que todas las instrucciones sehayan ejecutado al menos una vez. Lacomplejidad ciclomática se calcula mediante las tres formas siguientes:
   - **V (G) = E – N + 2**, donde E es el número de aristas del grafo de flujo y N es el número de nodos del mismo.
     En este caso **V (G) = 17 aristas – 14 nodos + 2 = 5.**
   - **V (G) = P + 1**, donde P es el número de nodos predicados[5] contenidos en el gráfico de flujo(Pressman, 2010).
     En este caso **V (G) = 4 nodos predicado + 1 = 5.**
   - **V (G) =R,** donde R es el número deregiones[6].
     En este caso **V (G) = 5 regiones.**

[5]Cada nodo que contiene una condición y está caracterizado porque dos o más aristas emergen de él.

[6] Espacios encerrados entre nodos y aristas

3. **Determinar un conjunto básico de rutas linealmente independientes**: el valor de V (G) indica el número de rutas linealmente independientes de la estructura de control del programa, por lo que se definen los 5 caminos obtenidos:

- **Ruta Básica 1:** 1-2-3-5-6-8-9-11-12-14
- **Ruta Básica 2:** 1-2-3-5-6-8-9-10-12-14
- **Ruta Básica 3:** 1-2-3-5-6-7-9-11-12-14
- **Ruta Básica 4:** 1-2-3-4-14
- **Ruta Básica 5:** 1-13-14

4. **Obtención de casos de pruebas:** cada ruta independiente es un caso de prueba a realizar, de forma que los datos introducidos provoquen que se visiten las sentencias vinculadas a cada nodo del camino. En este caso se obtuvieron 5 caminos básicos, que dan lugar a la confección de igual número de casos de pruebas, para aplicar las pruebas a este método. A continuación, se muestran los casos de pruebas.

Tabla 10. Caso de prueba de caja blanca para el camino básico #1.

Caso de prueba Camino Básico #1	
Descripción: este camino permite recibir un mensaje y guardarlo exitosamente en la base de datos. Luego se verifica si existe una conversación previa entre el usuario creador del mensaje y el usuario receptor, donde el usuario creador sea el propietario de esta conversación. Al ser así solamente se asocia el mensaje entrante a la conversación existente. Luego se verifica la misma condición para el caso donde el usuario receptor del mensaje sea el propietario de la conversación, y al cumplirse simplemente se asocia el mensaje entrante a dicha conversación.	
Entrada	El mensaje que envía el usuario
Resultado	Se registra el mensaje en la base de datos y se asocia a sus respectivas conversaciones.
Condiciones	Que no existan errores al guardar el objeto mensaje en la base de datos. Que existan conversaciones anteriores entre el creador y el receptor del mensaje entrante.

Fuente: elaboración propia

Tabla 11. Caso de prueba de caja blanca para el camino básico #2.

Caso de prueba Camino Básico #2
Descripción: este camino permite recibir un mensaje y guardarlo exitosamente en la base de datos. Luego se verifica si existe una conversación previa entre el usuario creador del mensaje y el usuario receptor, donde el usuario creador sea el

propietario de dicha conversación. Al ser así solamente se asocia el mensaje entrante a la conversación existente. Luego se verifica la misma condición para el caso donde el usuario receptor del mensaje sea el propietario de la conversación, y al no cumplirse simplemente se crea esta conversación, asociando implícitamente el mensaje entrante a la misma.	
Entrada	El mensaje que crea el usuario
Resultado	Se registra el mensaje en la base de datos y se asocia a sus respectivas conversaciones.
Condiciones	Que no existan errores al guardar el objeto mensaje en la base de datos. Que no exista una conversación previa donde el usuario receptor del mensaje sea el propietario.

Fuente: elaboración propia

Tabla 12. Caso de prueba de caja blanca para el camino básico #3.

Caso de prueba Camino Básico #3	
Descripción: este camino permite recibir un mensaje y guardarlo exitosamente en la base de datos. Luego se verifica si existe una conversación previa entre el usuario creador del mensaje y el usuario receptor, donde el usuario creador sea el propietario de dicha conversación. Al no ser así se crea dicha conversación, asociando implícitamente el mensaje entrante a la misma. Luego se verifica la misma condición para el caso donde el usuario receptor del mensaje sea el propietario de la conversación, y al cumplirse simplemente se asocia el mensaje entrante a dicha conversación.	
Entrada	El mensaje que envía el usuario
Resultado	Se registra el mensaje en la base de datos y se asocia a sus respectivas conversaciones.
Condiciones	Que no existan errores al guardar el objeto mensaje en la base de datos. Que no exista una conversación previa donde el usuario creador del mensaje sea el propietario.

Fuente: elaboración propia

Tabla 13. Caso de prueba de caja blanca para el camino básico #4.

Caso de prueba Camino Básico #4
Descripción: este camino permite recibir un mensaje enviado por un usuario y ejecutar una consulta de persistencia del objeto recibido a la base de datos. Luego se comprueba el éxito de dicha operación a través de una estructura condicional, sin embargo, al comprobar que la consulta a la base de datos no tuvo éxito se

retorna inmediatamente el resultado del método.	
Entrada	El mensaje que crea el usuario
Resultado	No se registra con éxito el mensaje enviado por el usuario.
Condiciones	Que existan errores al guardar el objeto mensaje en la base de datos.

Fuente: elaboración propia

Tabla 14. Caso de prueba de caja blanca para el camino básico #5.

Caso de prueba Camino Básico #5	
Descripción: este camino permite (a través de la estructura *try*)ejecutar la lógica del método supervisando la ocurrencia de una Excepción en el código, ya sea en tiempo de compilación o tiempo de ejecución. En el caso de prueba en cuestión una Excepción tiene lugar durante la ejecución del código los cual es manejado por la estructura *catch*, retornando inmediatamente el resultado del método.	
Entrada	El mensaje que crea el usuario
Resultado	No se registra con éxito el mensaje enviado por el usuario.
Condiciones	Que ocurra una Excepción durante la ejecución o compilación del método.

Fuente: elaboración propia

Descripción de la ejecución de los casos de prueba

Se realizaron pruebas manuales a cada caso de prueba simulando posibles situaciones a las cuales se pueda ser sometido el método a ejecutar. En los casos de prueba 1,2 y 3 se envió al sistema un mensaje bajo las siguientes cuatro combinaciones relativas a la existencia o no de conversaciones entre el usuario que envía el mensaje y aquel que lo recibe:

- Existen conversaciones donde ambos, tanto el creador como el receptor del mensaje son propietarios.
- No existen conversaciones previas registradas donde, tanto el creador como el receptor del mensaje sean propietarios.
- Existe una conversación donde el creador del mensaje es propietario, sin embargo, no existe una conversación donde el receptor del mensaje sea propietario.
- Existe una conversación donde el receptor del mensaje es propietario, sin embargo, no existe una conversación donde el creador del mensaje sea propietario.

En el caso de prueba 4 se envió un mensaje al sistema bajo la premisa de no contar con ningún servicio de persistencia de datos. Por último, en el caso de prueba 5 se provocó intencionalmente una Excepción interna del método, demostrando así la efectividad del manejo de excepciones.

Una vez ejecutados todos los casos de pruebas obtenidos con la técnica empleada, se concluye que los mismos fueron probados satisfactoriamente, corrigiéndose los hallazgos surgidos en una primera iteración y comprobándose su corrección en una segunda. Al concluir la prueba sedemuestra que todas las funcionalidadesde los componentesse ejecutan satisfactoriamente, quedando libres de código repetido o innecesario.

3.4 Conclusiones parciales

El uso de métricas de validación del diseño certifica la obtención de una solución flexible al mantenimiento y a la introducción de cambios. La realización de pruebas internas a nivel de unidad, con el empleo del método de caja blanca y la técnica camino básico, certifican la correctitud del código de la solución.

CONCLUSIONES GENERALES

- El estudio de los referentes teóricos vinculados con soluciones informáticas para la gestión de alertas, mensajes y notificaciones, permitió incorporar algunas de sus características a la solución propuesta.
- El empleo de técnicas de captura de requisitos, patrones de diseño y arquitectónicos, así como el uso de estándares de codificación en la implementación de la solución propuesta, permitió obtener una solución, flexible al mantenimiento y a la introducción de cambios.
- El desarrollo de pruebas de internas en el nivel de unidadpermitió corroborarla correcta organización y funcionamiento del código implementado, así como la depuración de este a través de la eliminación de código innecesario o repetido.

RECOMENDACIONES

- Concluir la implementación del *frontend* de los componentes propuestos.
- Integrar los componentes propuestos al producto XABAL-SAGAP 1.0

BIBLIOGRAFÍA REFERENCIADA

1. Programación de Interfaces de Usuario. (2018).
2. The PostgreSQL Global Development Group. (2019).
3. 1028-4788, R. M. (2019). Sistemas de notificaciones y alertas. *Mujeres*, 1.
4. Alegsa. (2020). *Alegsa*. Retrieved fromhttp://www.alegsa.com.ar/Dic/framework.php
5. Angular. (2020). *Angular*. Retrieved from https://angular.io/features
6. APROWEB, S. (2020). *softwareseleccion*. Retrieved from https://www.softwareseleccion.com/aproweb-p-965
7. ASP.NET, S. d. (2017). *MVC 3 Framework*. Retrieved from MVC 3 Framework: https://si.ua.es/es/documentacion/asp-net-mvc-3/1-dia/modelo-vista-controlador-mvc.html
8. Ayala Catari, I., Romero Marca, Y. T., & Serrano Urzagaste, R. J. (2010). *Estudio de herramientas CASE de soporte UML y UML2.*Retrieved from Scribd.com: http://es.scribd.com/doc/25374125/Estudio-de-Herramientas-CASE-de-Soporte-a-UML-y-UML2
9. Belkis de la C. García García, J. L. (2008). *SISGQR: SISTEMA DE GESTIÓN DE QUEJAS Y REPORTES DEL.* Matanzas.
10. Bembibre, C. (2009, Julio). *Definicion ABC*. Retrieved from https://www.definicionabc.com/comunicacion/mensaje.php
11. Bootstrap. (2020). *getbootstrap*.
12. Chiu, C. C. (2015). *Las pruebas en el desarrollo de software.* Mexico.
13. Enríquez Ruiz, J., Farías Palacín, E., & Flores Flores, E. (2017). *Metodología de desarrollo de software.* Universidad Católica Los Ángeles - Chimbote, Rectorado,

Chimbote - Perú. Retrieved from https://www.google.com.cu/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=2ahUKEwj80fWHxYXiAhURQq0KHY2LD4wQFjAAegQIARAC&url=https%3A%2F%2Fwww.uladech.edu.pe%2Fimages%2Fstories%2Funiversidad%2Fdocumentos%2F2018%2Fmetodologia-desarrollo-software-v001.pdf&usg=A

14. Gallegos, J. A. (2017). *Aplicación para dispositivos móviles mediante la utilización de frameworks con una arquitectura n-capas para la gestión de quejas ciudadanas integrado con el sistema de gestión documental quipux del GADM PASTAZA.*Puyo.
15. Gardey, J. P. (2013). *Definicion.de*. Retrieved from https://definicion.de/notificacion/
16. Hommel, S. (2019). *Estandares_de_codificacion_para_Java.*
17. JetBrains. (2020). *JetBrains*. Retrieved from https://www.jetbrains.com/idea/
18. JetBrains. (2020). *JetBrains*. Retrieved from https://www.jetbrains.com/webstorm/
19. Larman, C. (2016). *UML y Patrones. Una introducción al análisis y diseño orientado a objetos y al proceso unificado.*(3ra ed.). Prentice-Hall.
20. Londoño, J. H. (2005, abril 6). *Blogger*. Retrieved from http://ing-sw.blogspot.com/2005/04/tipos-de-pruebas-de-software.html
21. López de Jiménez, R. E. (2015, Noviembre 18). Sistemas de georreferenciación a través de dispositivos móviles para denuncias y quejas ciudadanas. *Artículos de Revista (ITCA-FEPADE)*.
22. Lorenz, M., & Kidd, J. (1994). *Object-Oriented Software Metrics.* Prentice-Hall.
23. Lucid_Software_Inc. (2019). *Lucidchart*. Retrieved from https://www.lucidchart.com/pages/es

24. Martin Olivera, Y., & Zamora Sanchez, G. (2016). *ENTORNO DE DESARROLLO INTEGRADO LIBRE Y MULTIPLATAFORMA PARA DESARROLLAR SOFTWARE EDUCATIVO EN FORMATO MULTIMEDIA.*
25. Mozilla. (2019). *MDN web docs mozilla*. Retrieved from developer.mozilla.org/en-US/docs/Web/HTML
26. Mozilla. (2019). *MDN web docs mozilla*. Retrieved from developer.mozilla.org/en-US/docs/Web/CSS/CSS3
27. Mozilla. (2020). *Docs, MDN Web*. Retrieved from https://developer.mozilla.org/en-US/docs/Web/JavaScript
28. Oracle. (2020). *Oracle*. Retrieved from https://docs.oracle.com/en/java/
29. PostgreSQL. (2020). *PostgreSQL*. Retrieved from https://www.postgresql.org/
30. Pressman, R. S. (2002). *Ingeniería del Software. Un enfoque práctico. Quinta Edición.*Mc Graw Hill.
31. Rocha, R. (2018). *Java EE 8 Design Patterns and Best Practices.*Birmingham: Packt.
32. Rodríguez Sánchez, T. (2015). *Metodología de desarrollo para la Actividad productiva de la UCI.* La Habana: Universidad de las Ciencias Informáticas.
33. Rodríguez Sanchéz, T. (2015). *Metodología de desarrollo para la Actividad productiva UCI.* La Habana.
34. Rodríguez, Y. B. (2016). *SISTEMA PARA LA GESTIÓN DE QUEJAS EN LA DIRECCIÓN MUNICIPAL.* Isla de la Juventud.
35. Rojas Karel, F. (2018). *Componente para la obtención de información desde el Sistema de Gestión para la Atención a la Población.*Ciudad Habana.
36. Rouse, M. (2019). *TeachTarget*. Retrieved from Database management system (DBMS): https://searchsqlserver.techtarget.com/definition/database-management-system

37. Rouse, M. (2020). *Tech Target*. Retrieved from https://searchsoftwarequality.techtarget.com/definition/pattern?_ga=2.163285817.444905919.1599000278-903531755.1599000278
38. Serafin, M. M. (2018). *Programador Jr. de Aplicaciones ASP. NET MVC: Manual de Estudiante.*
39. Solving Ad Hoc. (2017, Diciembre). *¿Qué son las historias de usuario y su función en agilidad?*Retrieved from https://solvingadhoc.com/las-historias-usuario-funcion-agilidad/
40. Sommerville, I. (2011). *Ingeniería de Software, 9na Edición.*
41. Spring. (2020). *Spring*. Retrieved from https://spring.io/guides/gs/spring-boot/
42. Stopford, B. (2018). Designing Event-Driven Systems.
43. Typescript, Microsoft. (2019). *TypeScript*. Retrieved from www.typescriptlang.org
44. UA. (2019). *Universidad de Alicante*. Retrieved from https://si.ua.es/es/documentacion/asp-net-mvc-3/
45. Visual Paradigm. (2020). *Visual Paradigm*. Retrieved from https://www.visual-paradigm.com/features/
46. Visual-Paradigm. (2019). *Visual-Paradigm*. Retrieved from www.visual-paradigm.com/solution/freeumltool/

ANEXOS

Anexo 1: acuerdos tomados en la tormenta de ideas con el cliente.

Tabla 15. Acuerdos tomados en la tormenta de ideas con el cliente.

Nº	Acuerdo	Responsable	Fecha
1	Las alertas y notificaciones deben generarse de manera automática atendiendo al comportamiento de los indicadores y los eventos del sistema.	Miguel Alejandro Diaz Rivera	20/1/2020
2	Los indicadores deben generarse de manera automática según las acciones que realicen los usuarios en el sistema.	Miguel Alejandro Diaz Rivera	20/1/2020
3	El usuario podrá consultar las alertas y notificaciones recibidas con anterioridad.	Miguel Alejandro Diaz Rivera	20/1/2020
4	Los mensajes solo pueden tener texto plano en su contenido.	Rafael Alejandro Aguilera	20/1/2020
5	El sistema podrá detectar de manera automática el uso de palabras o frases indebidas entre usuarios a través de los mensajes.	Rafael Alejandro Aguilera	20/1/2020
6	El usuario podrá enviar o compartir un mensaje a varios usuarios a la vez.	Rafael Alejandro Aguilera	20/1/2020
7	El usuario podrá consultar los mensajes intercambiados con anterioridad con otros usuarios.	Rafael Alejandro Aguilera	20/1/2020

Fuente: elaboración propia

Anexo 2: preguntas realizadas en la entrevista con el cliente.

A continuación, se enumeran las preguntas realizadas durante la entrevista con el cliente.

1- ¿Qué elementos del negocio debe tener en cuenta el sistema a la hora de generar una notificación?
2- ¿Qué elementos del negocio debe tener en cuenta el sistema a la hora de generar una alerta?
3- ¿Qué tipo y formato de contenido debe permitir compartir el sistema a través de los mensajes?
4- ¿Puede el usuario consultar las notificaciones y alertas que ha recibido con anterioridad?
5- ¿Debe el usuario poder eliminar las alertas y notificaciones recibidas con anterioridad?
6- ¿Debe un usuario poder generar y enviar una alerta o notificación a otro usuario?

Anexo 3: historias de usuario.

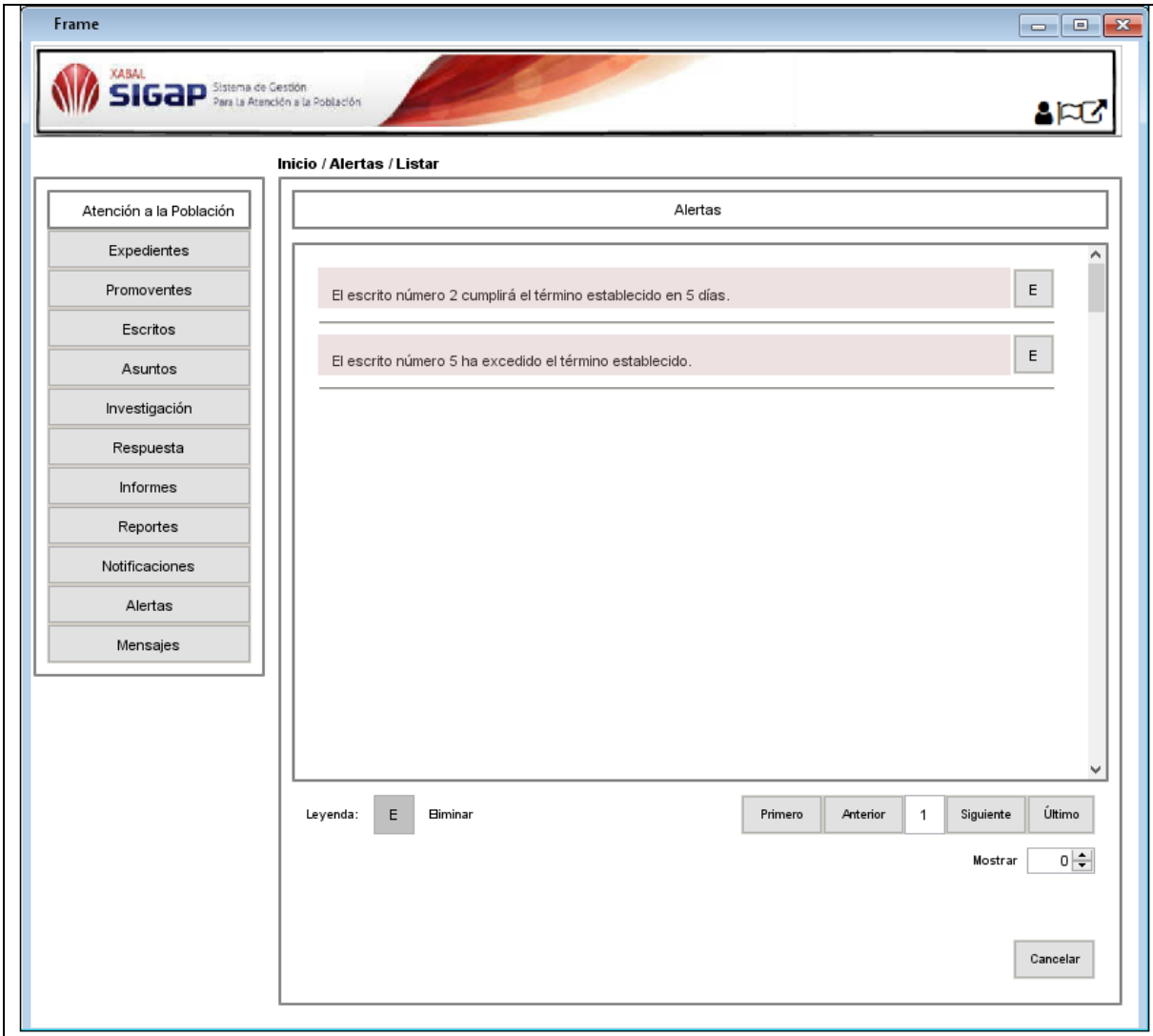
Tabla 16. Historia de usuario Generar alerta.

Historias de Usuario	
Número: 1	**Nombre de la HU:** Generar alerta.
Programador: Miguel Alejandro Diaz Rivera	**Iteración asignada:** 1
Prioridad: alta	**Tiempo estimado:** 8 días
Riesgo en desarrollo: alto	**Tiempo real:** 8 días
Descripción: el sistema genera y registra alertas basadas en indicadores y las envía al usuario destinatario, teniendo en cuenta la ocurrencia de eventos en el sistema.	
Observaciones: N/A	
Prototipo de interfaz: N/A	

Fuente: elaboración propia.

Tabla 17. Historia de usuario Listar alerta.

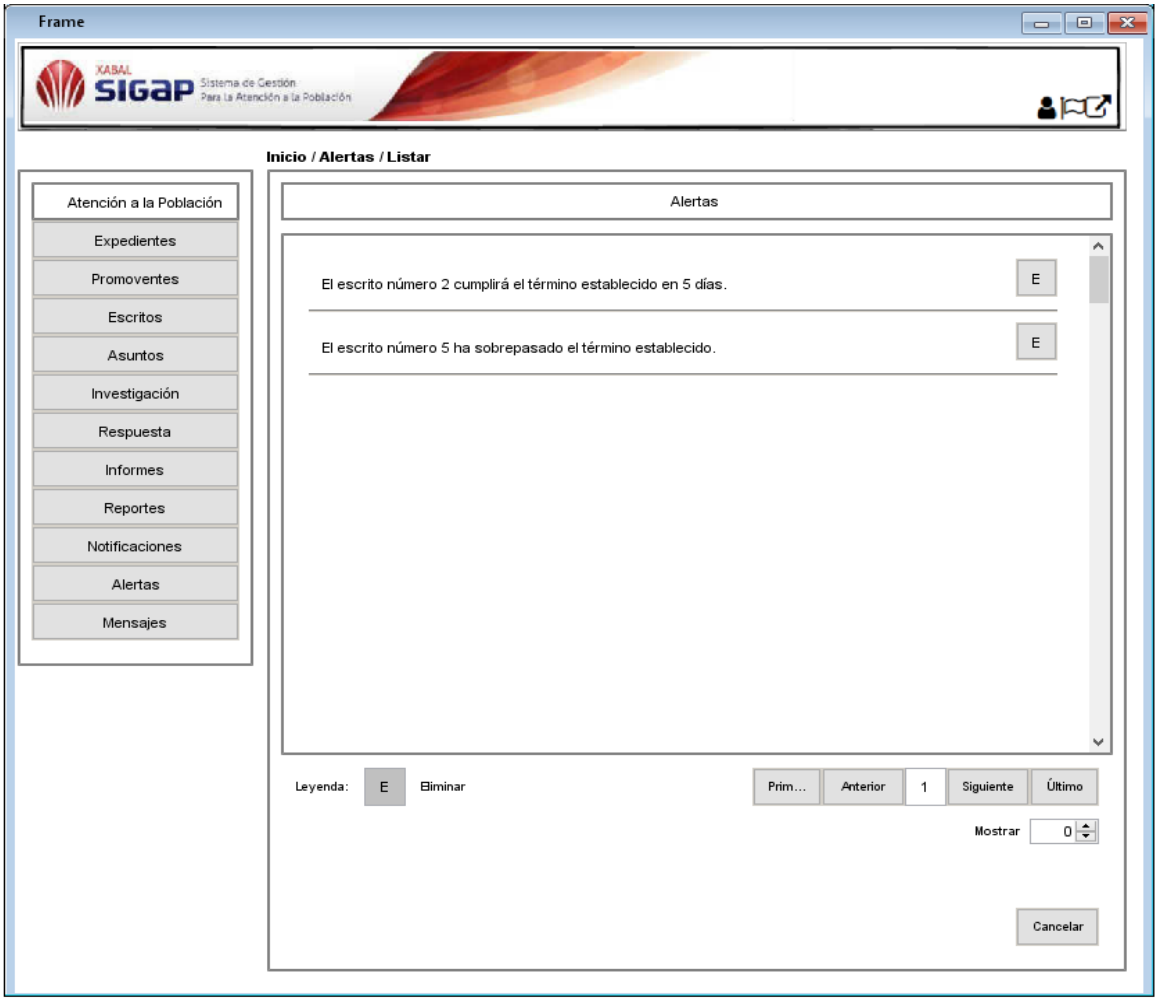
Historias de Usuario	
Número: 2	**Nombre de la HU:** Listar alerta.
Programador: Rafael Alejandro Aguilera Rodríguez	**Iteración asignada:** 1
Prioridad: alta	**Tiempo estimado:** 5 días
Riesgo en desarrollo: alto	**Tiempo real:** 5 días
Descripción: permite a los usuarios del SIGAP observar las alertas que le son generadas por el sistema.	
Observaciones: N/A	
Prototipo de interfaz:	



Fuente: elaboración propia.

Tabla 18. Historia de usuario Cambiar estado de la alerta.

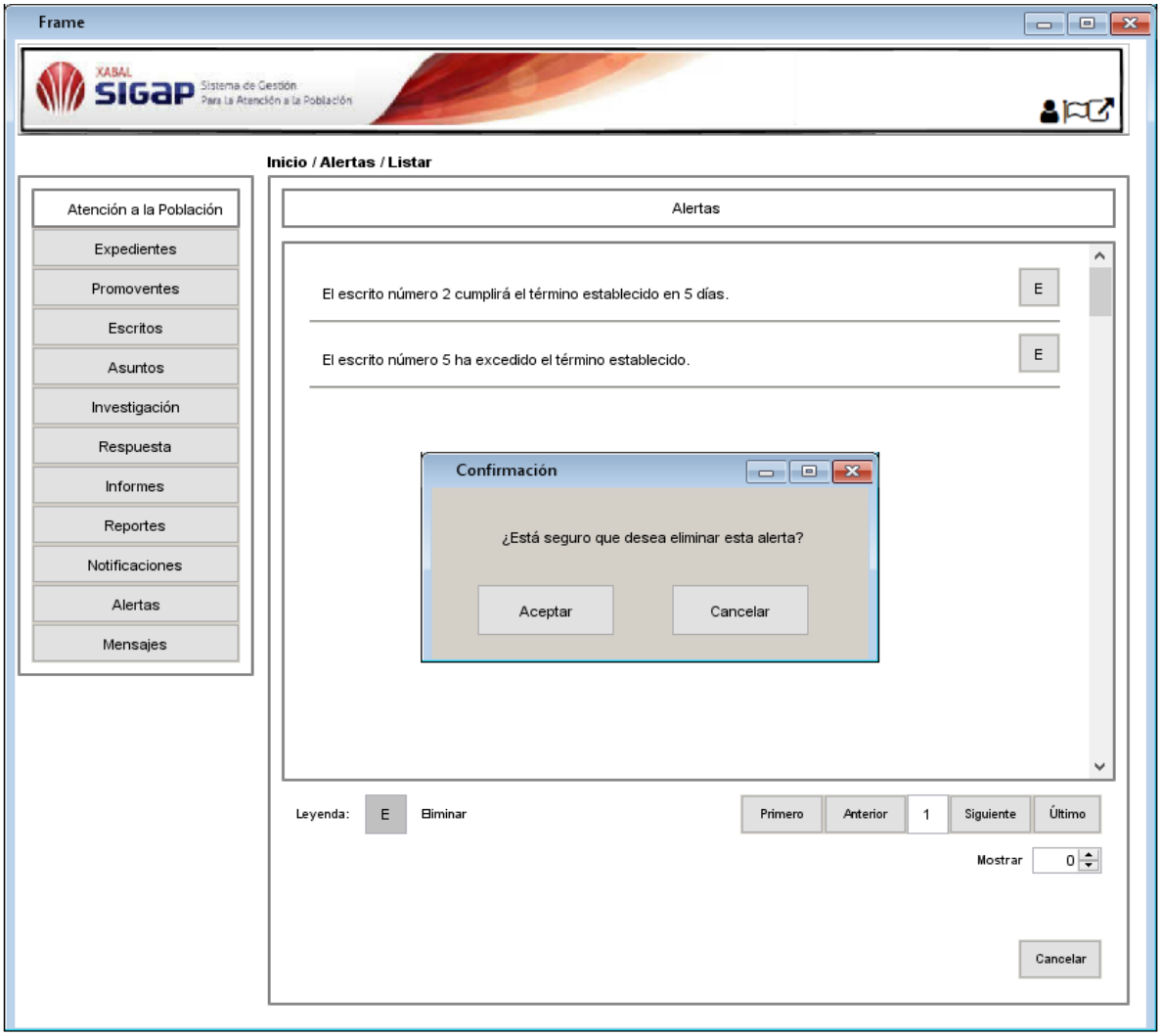
Historias de Usuario	
Número: 3	**Nombre de la HU:** Cambiar estado de la alerta.
Programador: Miguel Alejandro Diaz Rivera	**Iteración asignada:** 1
Prioridad: alta	**Tiempo estimado:** 8 días
Riesgo en desarrollo: alto	**Tiempo real:** 8 días
Descripción: una vez que los usuarios del SIGAP leen las alertas generadas, el sistema cambia el estado de las mismas de no leída al estado de leída.	

Observaciones: N/A
Prototipo de interfaz: 

Fuente: elaboración propia.

Tabla 19. Historia de usuario Eliminar alerta.

Historias de Usuario	
Número: 4	**Nombre de la HU:** Eliminar alerta.
Programador: Miguel Alejandro Diaz Rivera	**Iteración asignada:** 1
Prioridad: alta	**Tiempo estimado:** 8 días
Riesgo en desarrollo: alto	**Tiempo real:** 8 días
Descripción: permite al usuario eliminar una alerta recibida, removiéndola del panel de alertas.	

Observaciones: N/A
Prototipo de interfaz: 

Fuente: elaboración propia.

Tabla 20. Historia de usuario Generar notificación.

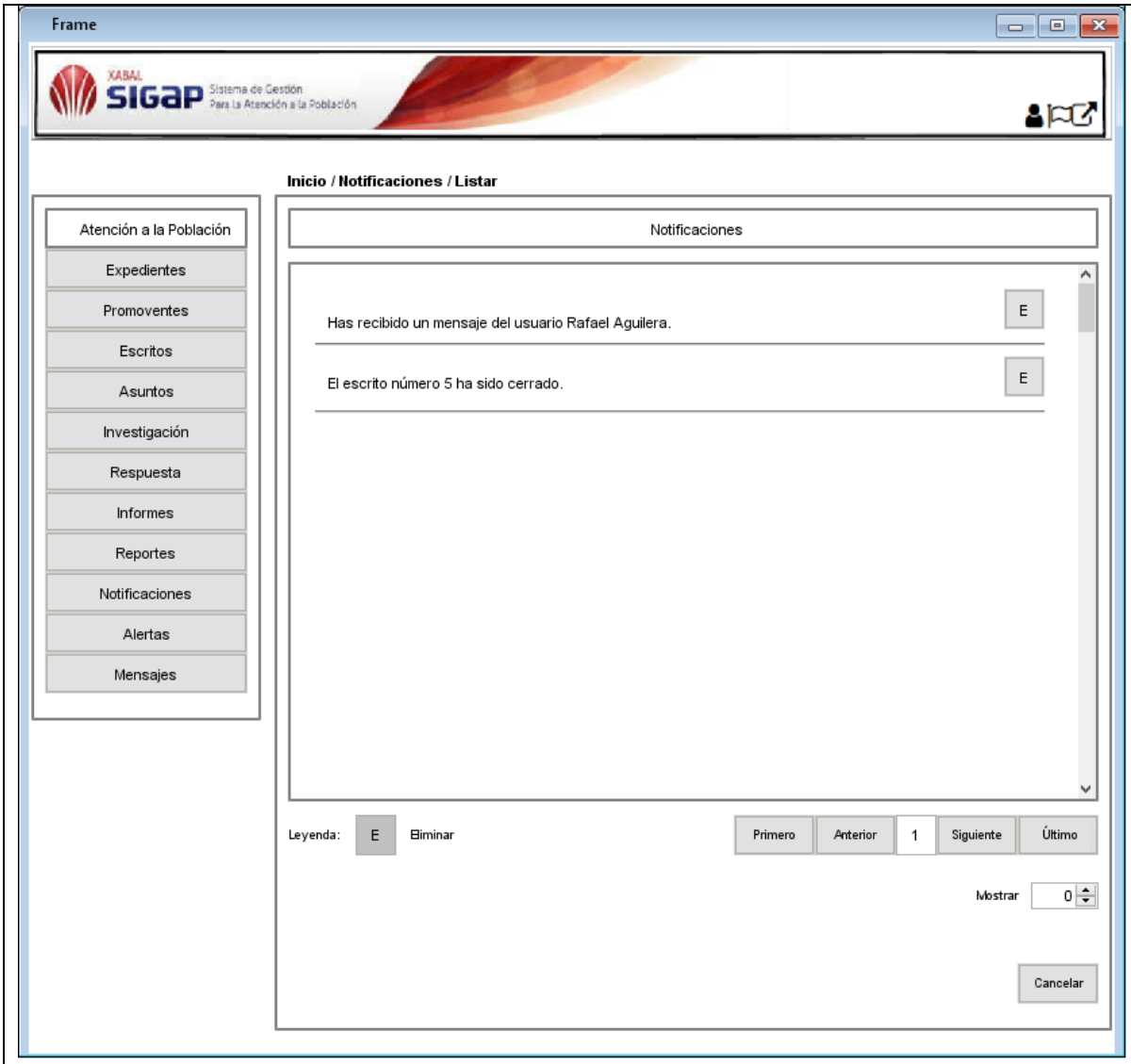
Historias de Usuario	
Número: 5	**Nombre de la HU:** Generar notificación.
Programador: Miguel Alejandro Diaz Rivera	**Iteración asignada:** 1
Prioridad: alta	**Tiempo estimado:** 11 días
Riesgo en desarrollo: alto	**Tiempo real:** 11 días

Descripción: permite al SIGAP crear y almacenar notificaciones a partir de los eventos que tienen lugar en el sistema y la información que estos proporcionan. Además, se tiene en cuenta el tipo de notificación a generar, según la información que se le debe proporcionar al usuario, así como los usuarios a los cuales va dirigida la notificación.
Observaciones: N/A
Prototipo de interfaz: N/A

Fuente: elaboración propia.

Tabla 21. Historia de usuario Cambiar estado de la notificación.

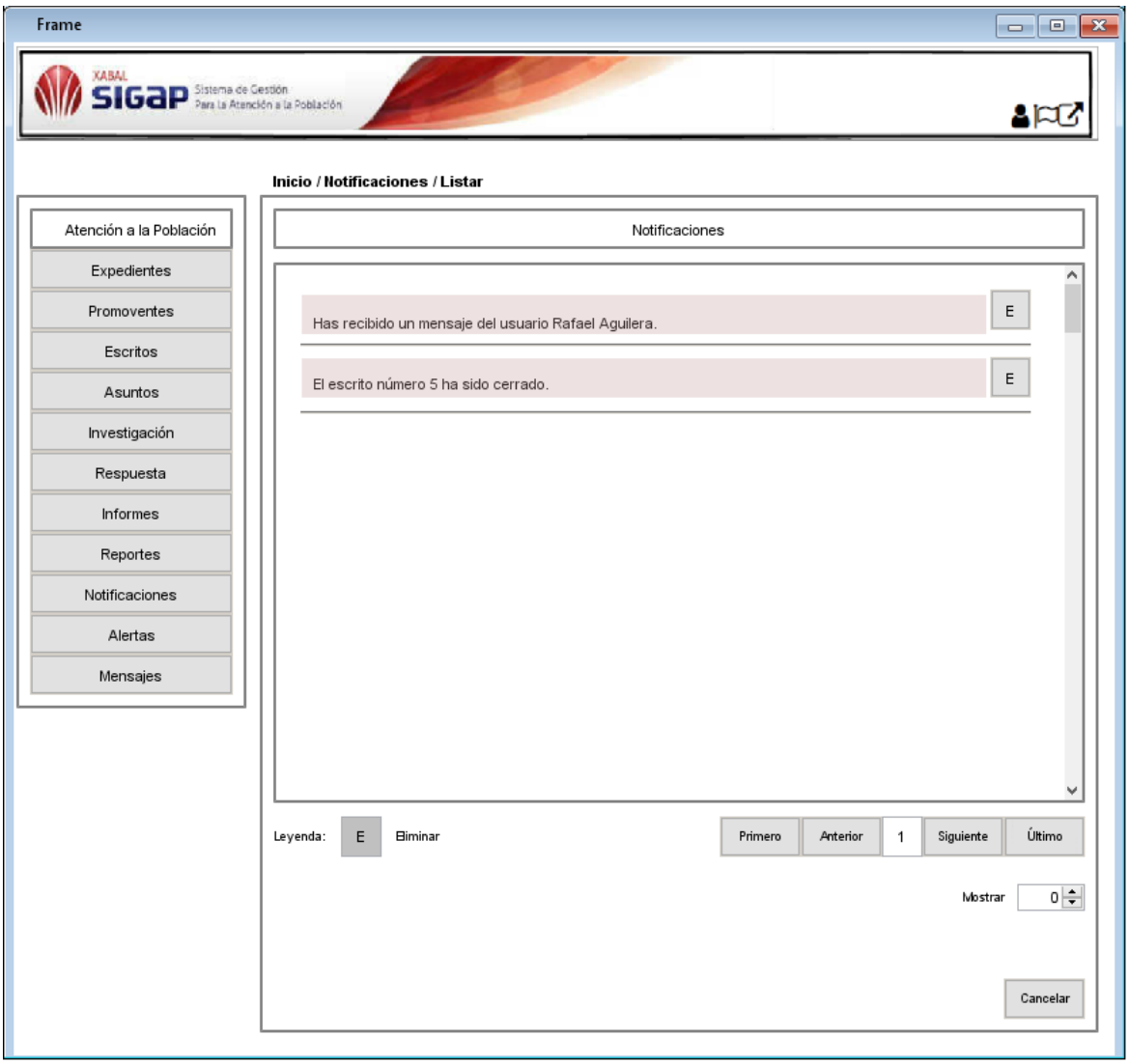
Historias de Usuario	
Número: 6	**Nombre de la HU:** Cambiar estado de la notificación.
Programador: Rafael Alejandro Aguilera Rodríguez	**Iteración asignada:** 1
Prioridad: alta	**Tiempo estimado:** 9 días
Riesgo en desarrollo: alto	**Tiempo real:** 9 días
Descripción: una vez que los usuarios del SIGAP leen las notificaciones enviadas por el sistema, el sistema las cambia del estado de no leída hacia el estado de leída.	
Observaciones: N/A	
Prototipo de interfaz:	



Fuente: elaboración propia.

Tabla 22. Historia de usuario Listar notificación.

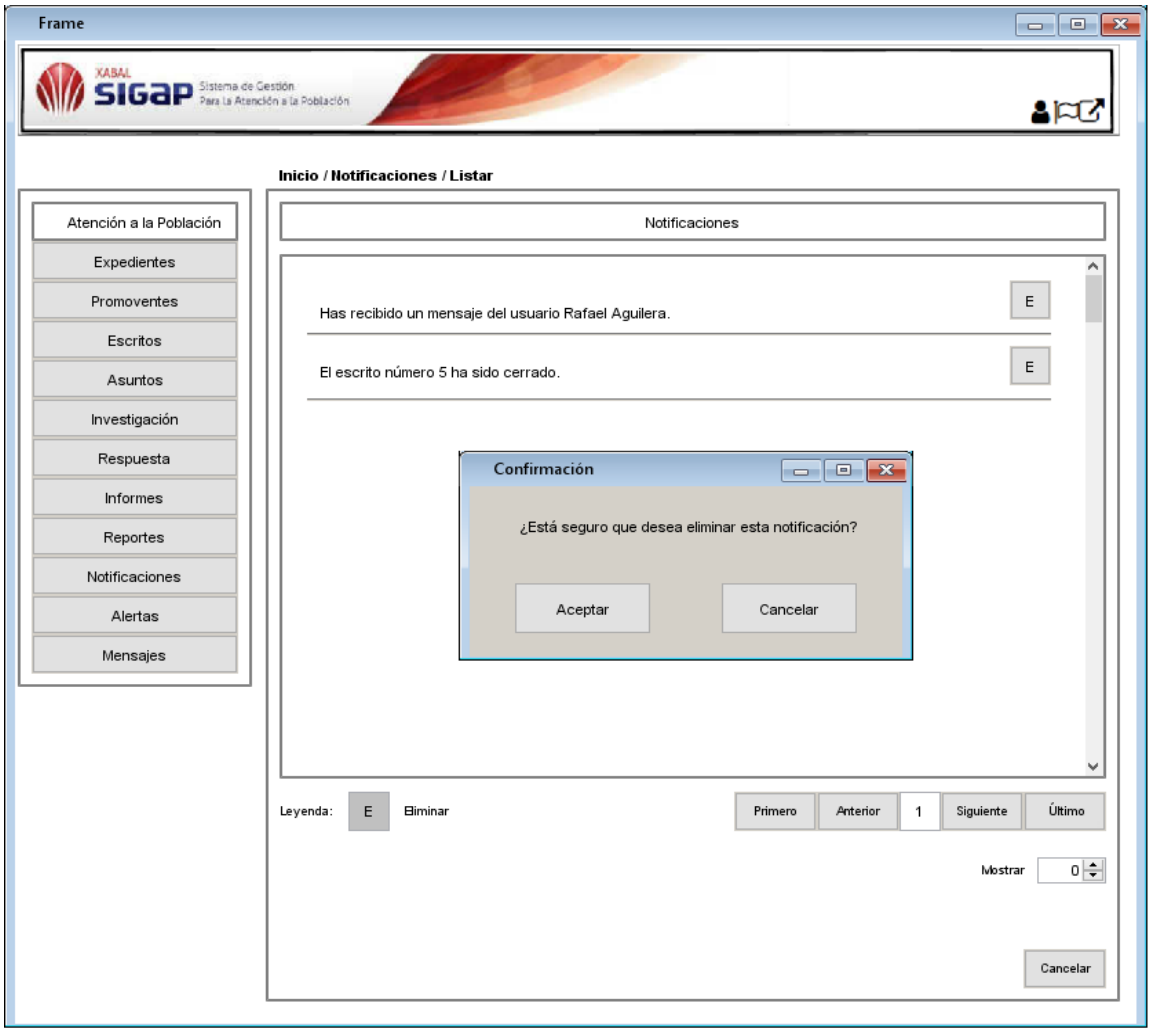
Historias de Usuario	
Número: 7	**Nombre de la HU:** Listar notificación.
Programador: Miguel Alejandro Díaz Rivera	**Iteración asignada:** 1
Prioridad: alta	**Tiempo estimado:** 6 días
Riesgo en desarrollo: alto	**Tiempo real:** 6 días
Descripción: permite a los usuarios del SIGAP observar las notificaciones que le son generadas por el sistema.	

Observaciones: N/A
Prototipo de interfaz: 

Fuente: elaboración propia.

Tabla 23. Historia de usuario Eliminar notificación.

Historias de Usuario	
Número: 8	**Nombre de la HU:** Eliminar notificación.
Programador: Miguel Alejandro Díaz Rivera	**Iteración asignada:** 1
Prioridad: alta	**Tiempo estimado:** 9 días

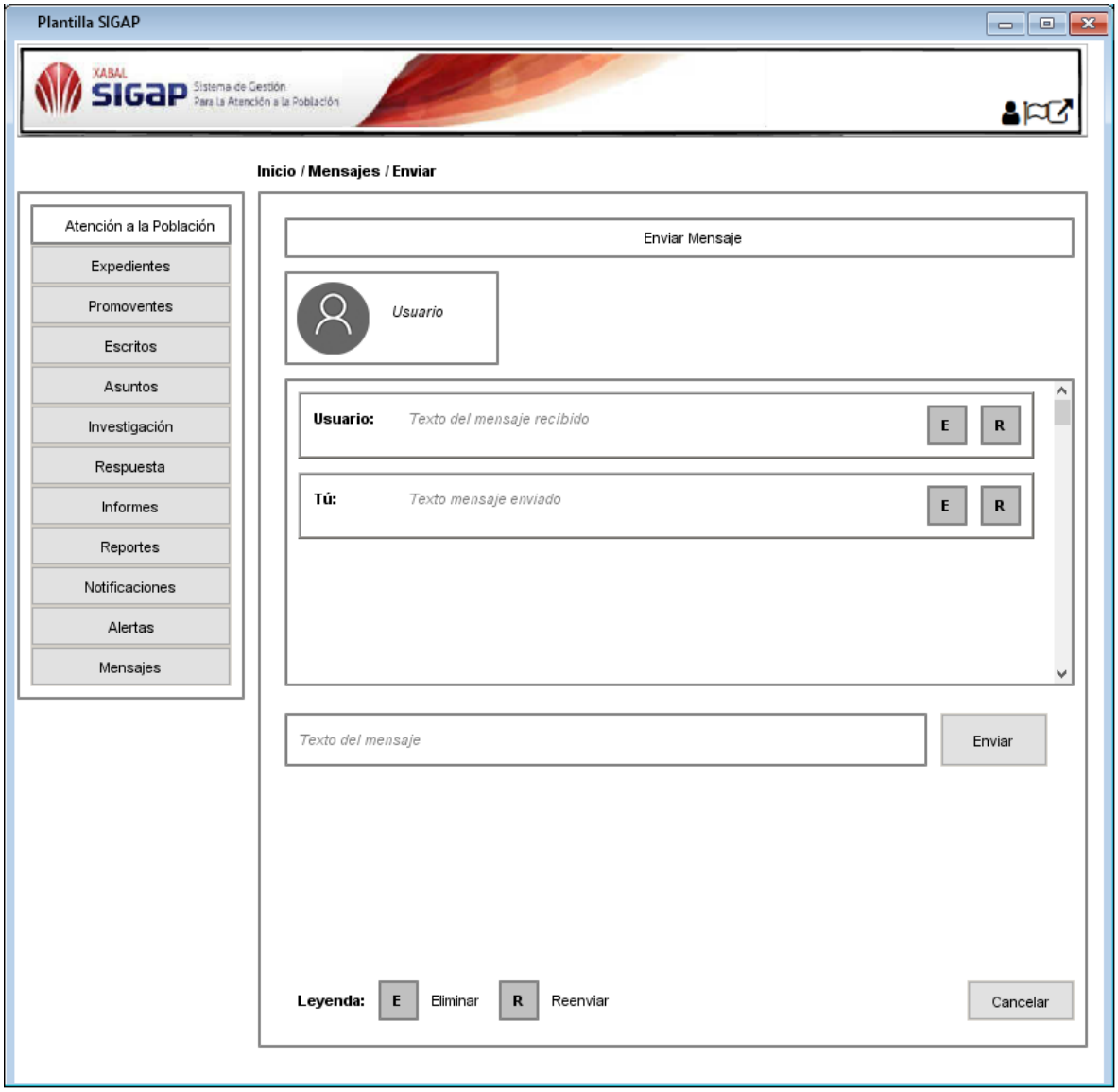
Riesgo en desarrollo: alto	**Tiempo real:** 9 días
Descripción: permite a un usuario eliminar una notificación recibida, removiéndola el panel de notificaciones.	
Observaciones: N/A	
Prototipo de interfaz: 	

Fuente: elaboración propia.

Tabla 24. Historia de usuario Enviar mensaje.

Historias de Usuario	
Número: 9	**Nombre de la HU:** Enviar mensaje.
Programador: Miguel Alejandro Díaz Rivera	**Iteración asignada:** 1

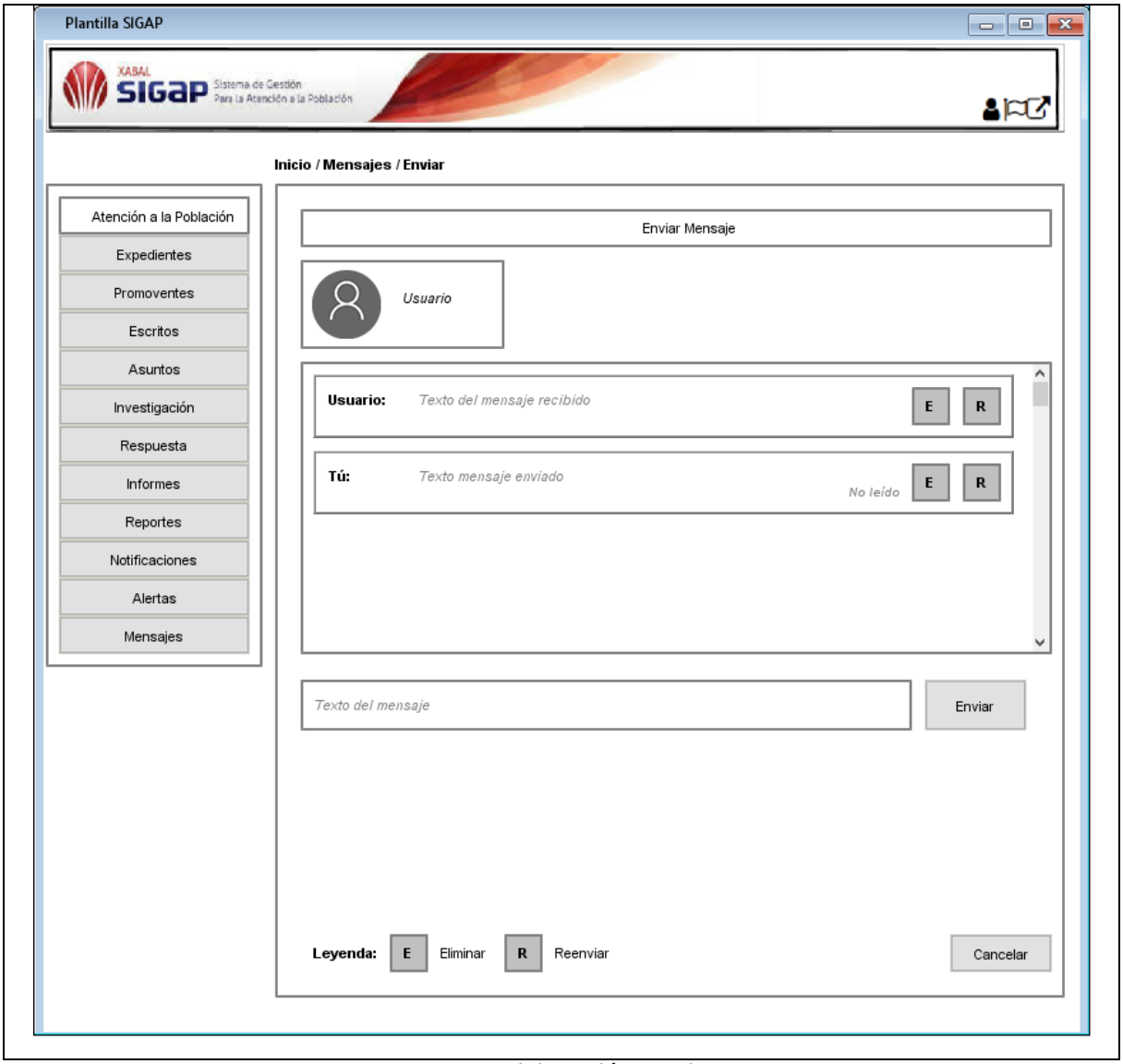
Prioridad: alta	**Tiempo estimado:** 9 días
Riesgo en desarrollo: alto	**Tiempo real:** 9 días
Descripción: permite a un usuario autenticado en el sistema, enviar un mensaje de texto a otro usuario registrado en el sistema. De esta manera permite al sistema registrar y almacenar estos mensajes en la base de datos, así como los datos vinculados a los mismos.	
Observaciones: N/A	
Prototipo de interfaz:	



Fuente: elaboración propia.

Tabla 25. Historia de usuario Listar mensajes.

Historias de Usuario	
Número: 10	**Nombre de la HU:** Listar Mensaje.
Programador: Miguel Alejandro Díaz Rivera	**Iteración asignada:** 1
Prioridad: alta	**Tiempo estimado:** 8 días
Riesgo en desarrollo: alto	**Tiempo real:** 8 días
Descripción: permite a los usuarios del SIGAP observar los mensajes que le entran a través del sistema	
Observaciones: N/A	
Prototipo de interfaz:	

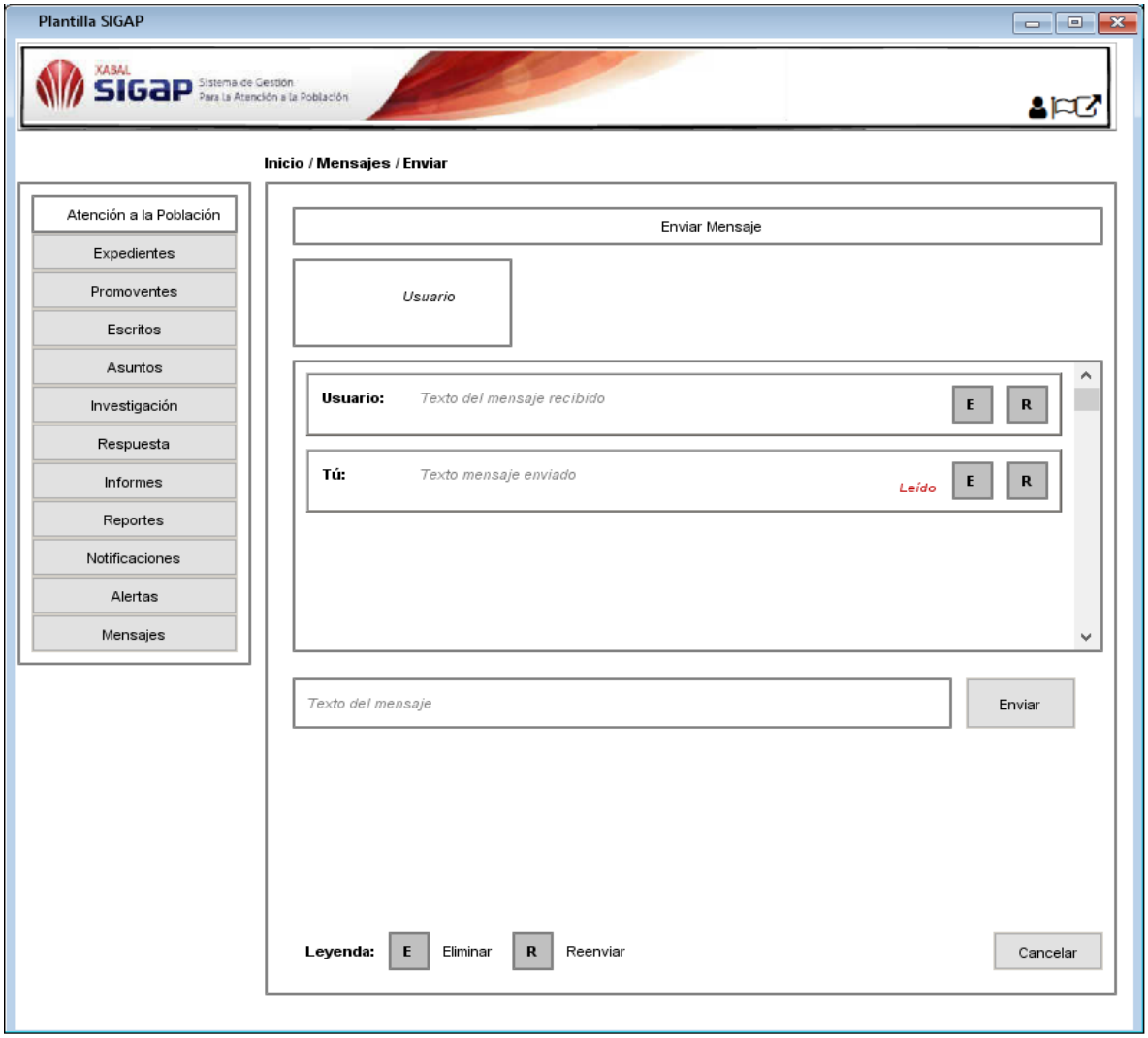


Fuente: elaboración propia.

Tabla 26. Historia de usuario Cambiar estado del mensaje.

Historias de Usuario	
Número: 11	**Nombre de la HU:** Cambiar estado del mensaje.
Programador: Rafael Alejandro Aguilera Rodríguez	**Iteración asignada:** 1
Prioridad: alta	**Tiempo estimado:** 10 días
Riesgo en desarrollo: alto atención extravío	**Tiempo real:** 10 días
Descripción: una vez que los usuarios del SIGAP leen los mensajes enviados por otros usuarios,	

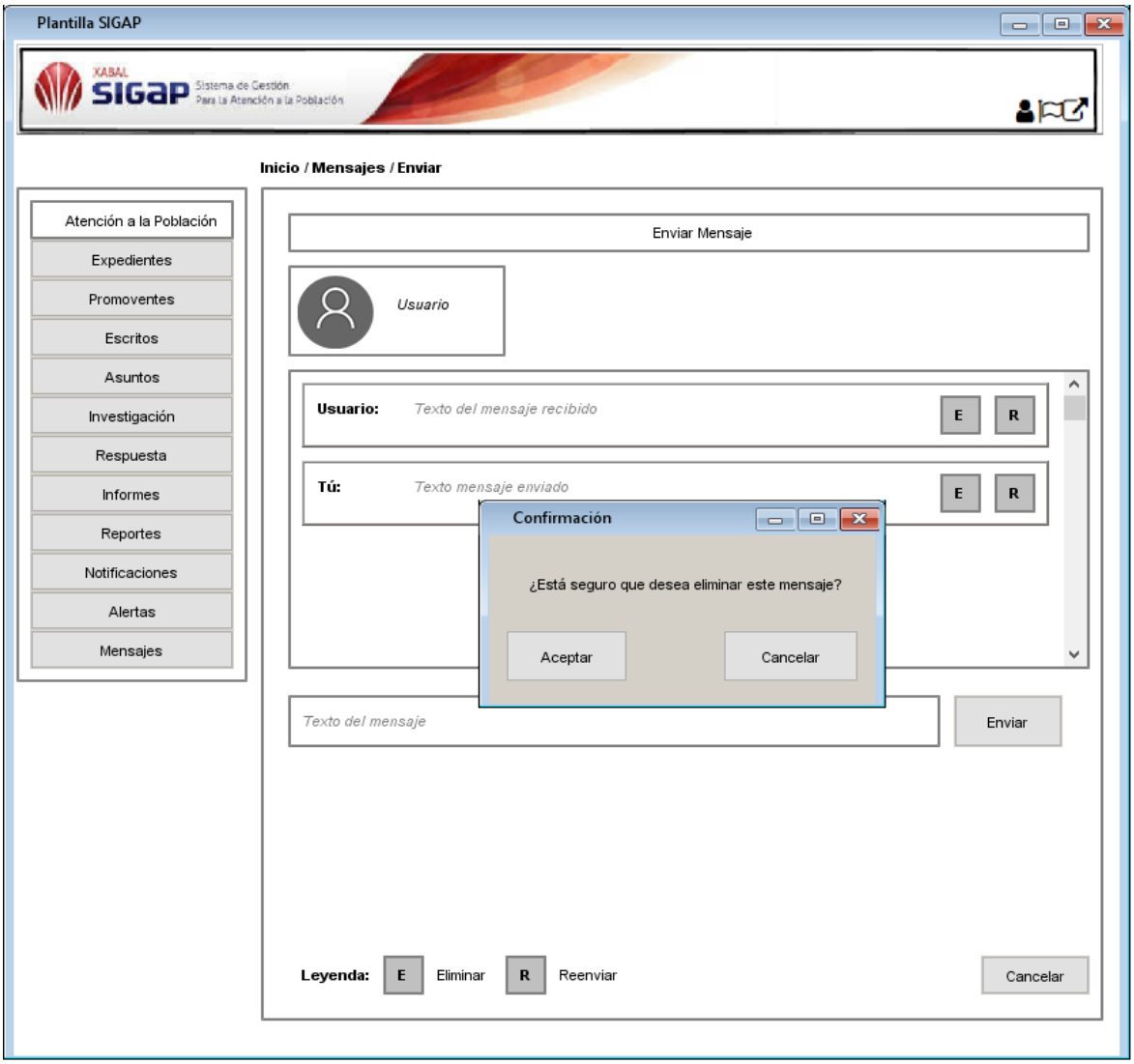
el sistema los cambia del estado de no leído hacia el estado de leído.
Observaciones: N/A
Prototipo de interfaz:



Fuente: elaboración propia.

Tabla 27. Historia de usuario Eliminar mensaje.

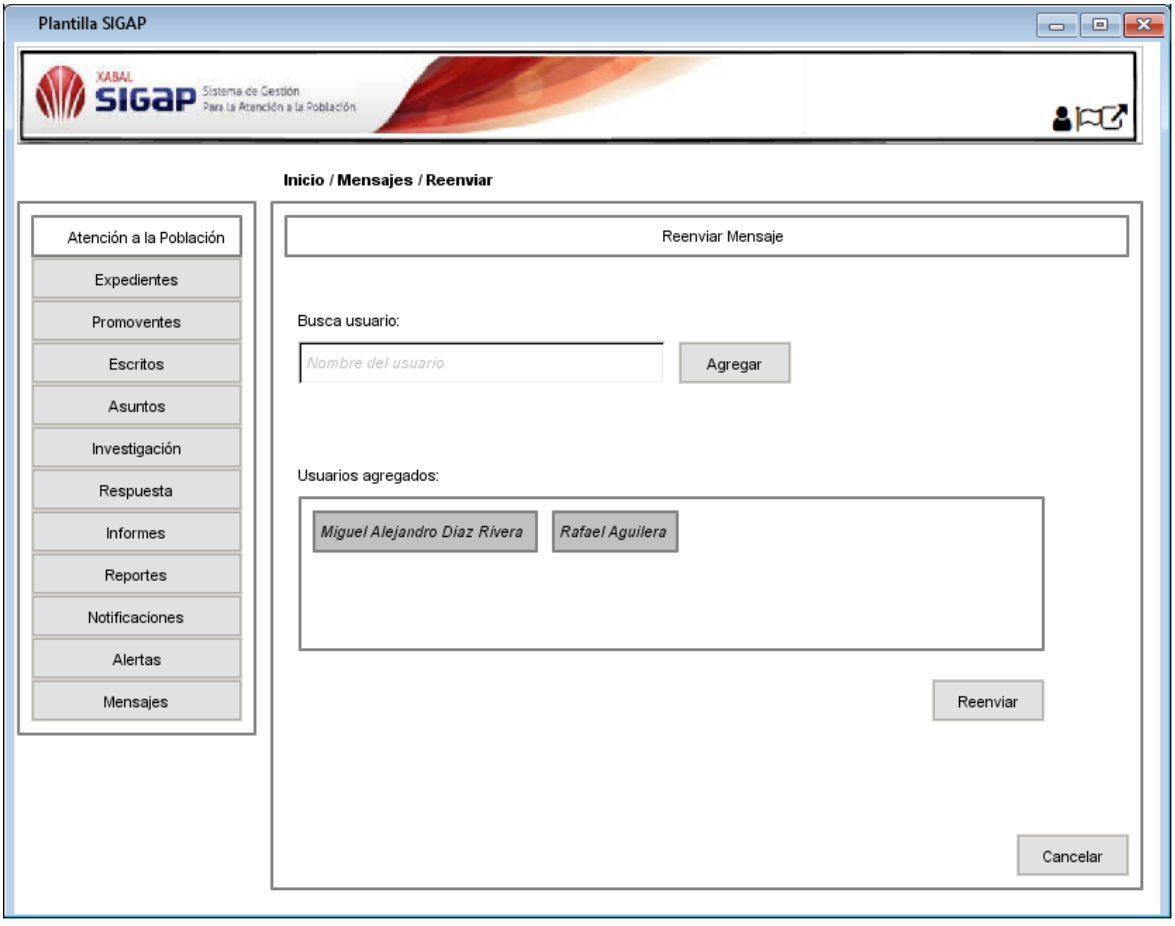
Historias de Usuario	
Número: 12	**Nombre de la HU:** Eliminar mensaje.
Programador: Rafael Alejandro Aguilera Rodríguez	**Iteración asignada:** 1
Prioridad: alta	**Tiempo estimado:** 8 días

Riesgo en desarrollo: alto	**Tiempo real:** 8 días
Descripción: permite a los usuarios del SIGAP eliminar un mensaje de una conversación con otro usuario.	
Observaciones: N/A	
Prototipo de interfaz: 	

Fuente: elaboración propia.

Tabla 28. Historia de usuario Reenviar mensaje

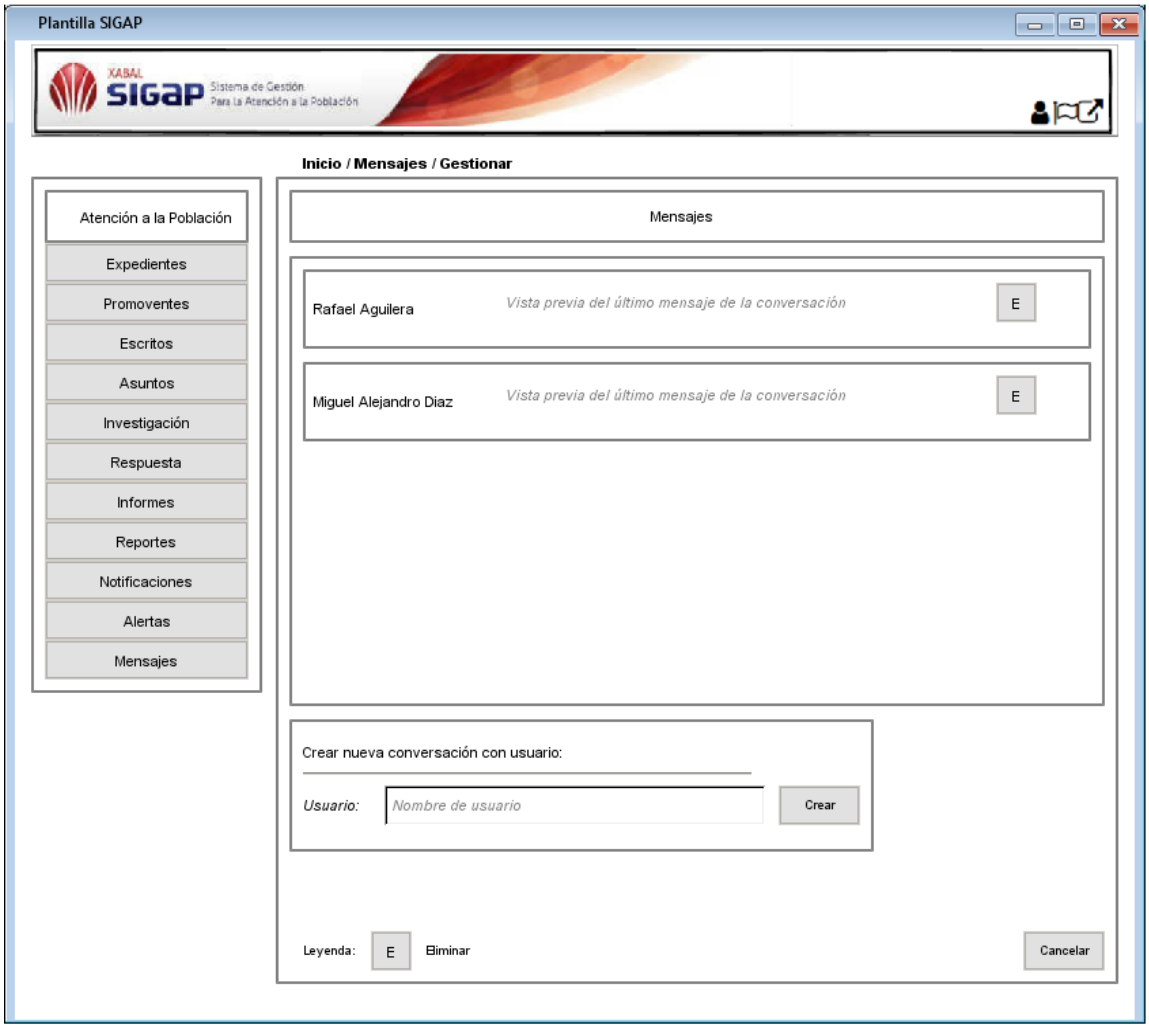
Historias de Usuario

Número: 13	**Nombre de la HU:** Reenviar mensaje.
Programador: Miguel Alejandro Díaz Rivera	**Iteración asignada:** 1
Prioridad: alta	**Tiempo estimado:** 8 días
Riesgo en desarrollo: alto	**Tiempo real:** 8 días
Descripción: permite a los usuarios del SIGAP seleccionar los mensajes de textos que hayan sido enviados y poder enviarlos nuevamente a otros usuarios, uno a la vez. El campo agregar permite añadir los usuarios a los cuales le llegará el mensaje.	
Observaciones: N/A	
Prototipo de interfaz: 	

Fuente: elaboración propia.

Tabla 29. Historia de usuario Crear Conversación.

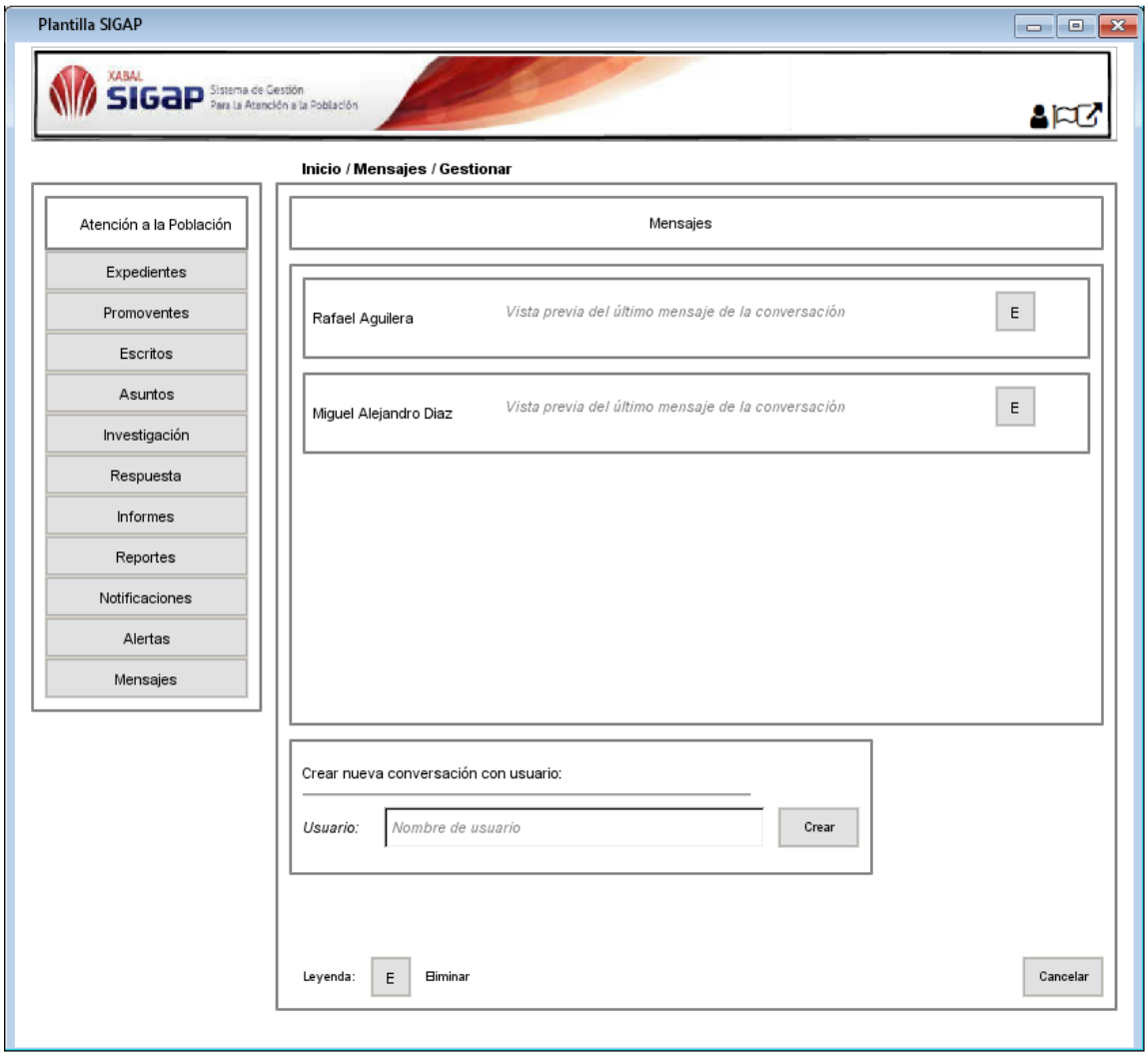
Historias de Usuario	
Número: 14	**Nombre de la HU:** Crear conversación.

Programador: Miguel Alejandro Díaz Rivera	**Iteración asignada:** 1
Prioridad: alta	**Tiempo estimado:** 6 días
Riesgo en desarrollo: alto	**Tiempo real:** 6 días
Descripción: permite a los usuarios del SIGAP iniciar una nueva conversación con otros usuarios registrados en el sistema.	
Observaciones: N/A	
Prototipo de interfaz: 	

Fuente: elaboración propia.

Tabla 30. Historia de usuario Listar conversación.

Historias de Usuario	
Número: 15	**Nombre de la HU:** Listar conversación.

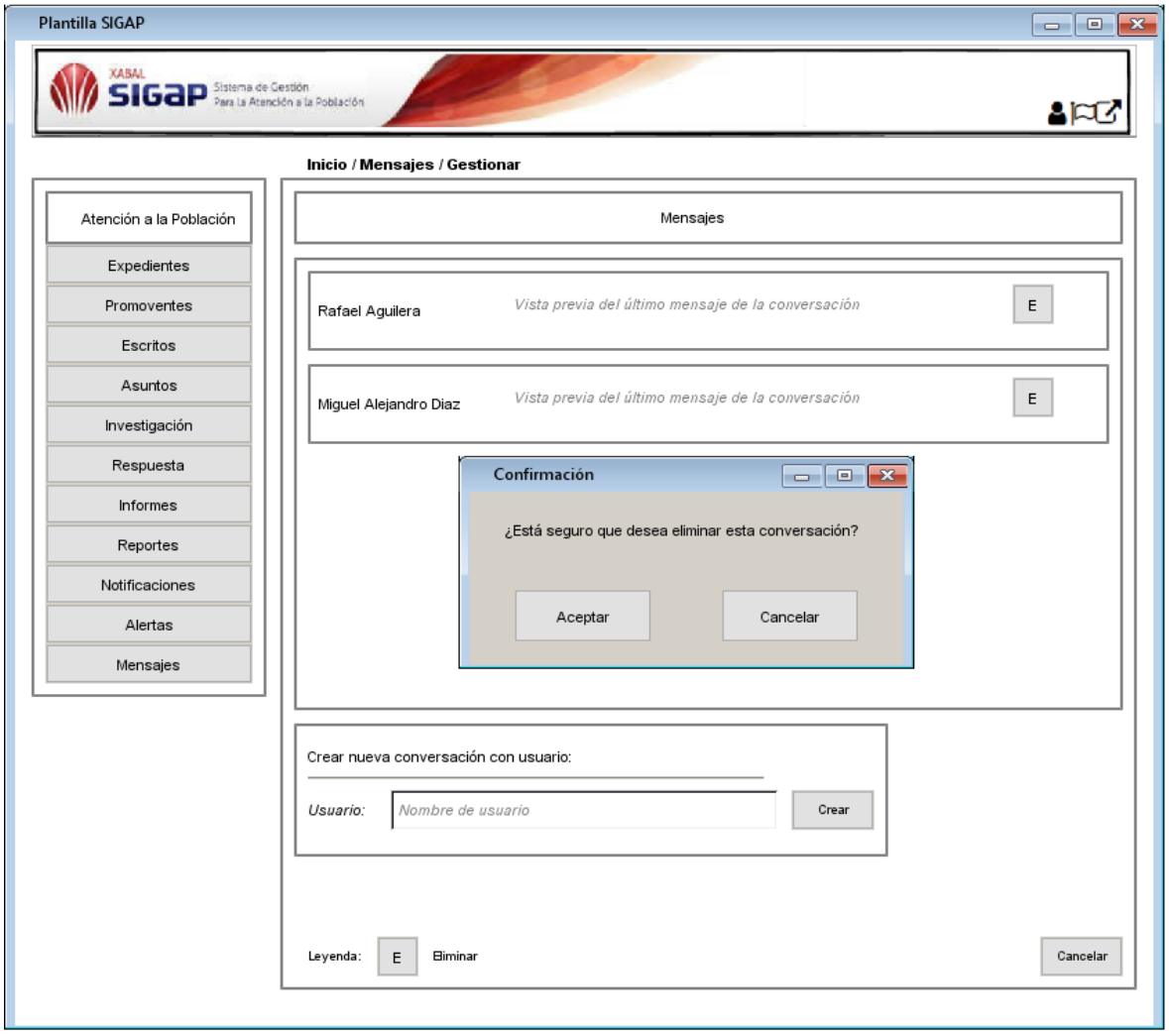
Programador: Miguel Alejandro Díaz Rivera	**Iteración asignada:** 1
Prioridad: alta	**Tiempo estimado:** 6 días
Riesgo en desarrollo: alto	**Tiempo real:** 6 días
Descripción: permite a los usuarios listar las conversaciones que han tenido previamente con otros usuarios del sistema, permitiéndoles gestionar sus mensajes con dichos usuarios.	
Observaciones: N/A	
Prototipo de interfaz: 	

Fuente: elaboración propia.

Tabla 31. Historia de usuario Eliminar conversación.

Historias de Usuario	
Número: 16	**Nombre de la HU:** Eliminar conversación.

Programador: Miguel Alejandro Díaz Rivera	**Iteración asignada:** 1
Prioridad: alta	**Tiempo estimado:** 6 días
Riesgo en desarrollo: alto	**Tiempo real:** 6 días
Descripción: permite a los usuarios eliminar el historial de mensajes que tienen con los usuarios del sistema mediante la eliminación de las conversaciones con dichos usuarios.	
Observaciones: N/A	
Prototipo de interfaz:	



Fuente: elaboración propia.