Universidad de las Ciencias Informáticas

Facultad 3

Componentes para la gestión de alertas y notificaciones en la herramienta informática Expediente Judicial Electrónico

---

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

Autor: Omar Sierra Sarduy

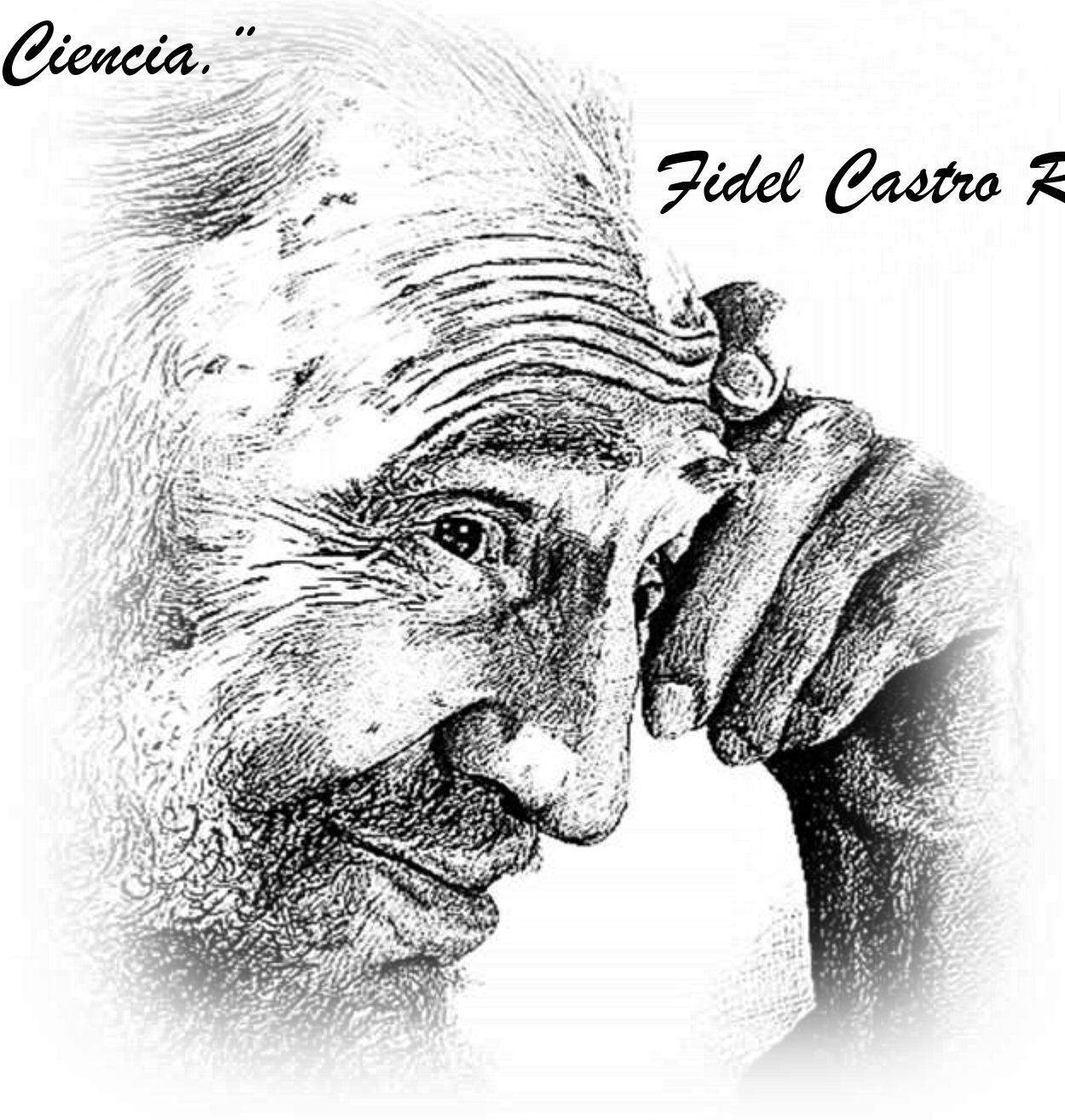
Tutor(es): Ing. Reinier Fernández Coello

Ing. Luis Ernesto Méndez Castillo

La Habana, 2019

"El futuro de nuestra Patria tiene que ser necesariamente un futuro de hombres de Ciencia."

Fidel Castro Ruz



DECLARACIÓN DE AUTORÍA

Declaro ser el único autor del presente trabajo de diploma que tiene por título: "*Componentes para la gestión de alertas y notificaciones en la herramienta informática Expediente Judicial Electrónico*" y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste se firma la presente a los ____ días del mes de _____ del año _____.

Firma del autor

Omar Sierra Sarduy

Firma del tutor

Ing. Reinier Fernández Coello

Firma del tutor

Ing. Luis Ernesto Méndez Castillo

AGRADECIMIENTOS

A mi madre por estar siempre a pesar de la distancia, ...mi razón de ser, la primera mujer en mi vida, esa flor que cuido en el patio y que no dejo que nadie toque. A mi padre por guiarme siempre por el camino correcto, ...mi mejor amigo, mi hermano, el mejor padre del mundo.

A mi abuela, la mujer maravilla, la razón por la que creo que siempre se puede y que nada es imposible, ella sabe cómo se hace lo difícil, una dama inteligente, la diosa de la sabiduría.

A toda mi familia, que me apoyó desde la distancia, ...cada regaño, cada llamada de atención, nada fue en vano, todo valió la pena, por ustedes he llegado tan lejos.

A mis tutores, ¡no tengo forma de agradecer por la ayuda brindada!, disculpen mis malcriadeces. Luisito perdón por las veces que ocupé tu tiempo libre de Gym.

A Reinier, gracias por ser incondicional, por estar cuando todo parecía imposible, es bueno saber que aún quedan personas en el mundo así, realmente un ejemplo a seguir. Gracias a todas las personas del proyecto por brindar su ayuda en cualquier duda.

A todos mis compañeros de estudio, a mis compañeros de apartamento, de ellos me quedo con los mejores momentos, las mejores bromas, los que siempre me acompañaban al comedor y en mis desamores:

Raiko, Alejandro, Yoandry (mi Yoyo), Yaniel (el cariñoso Hipo), Miguel (cariñosamente El machi), Francis, Yudiel, Jesus, Victor, Falcon, Noel, Yelina, Celida, Claudia, Elizabeth, Esmirna, Yulie, Mirna, la especial y apodada cariñosamente "bola de pelos" Yannia, compañera y amiga, no voy a olvidar tus bromas y llamadas telefónicas, cuando me sentía solo.

A mi pedacito de porcelana, como le llamo cuidadosamente: Haraid, ella que ha sido motivo de mis transformaciones en mi vida, por la que esperé tanto, gracias por estar siempre ahí, me lamento de no haberla conocido antes... nunca es tarde si la dicha es buena.

A todas las personas que me aconsejaron, y estuvieron a mi lado de alguna forma u otra durante todo el camino. A todos muchas gracias!

DEDICATORIA

Quiero dedicar esta tesis a todas las personas que han estado conmigo en las buenas y en las malas, en especial a mis padres por haber estado tan cerca cuando la distancia era tan grande.

RESUMEN

Los Tribunales Populares Cubanos han dado pasos de avance en el uso de las Tecnologías de la Información y las Comunicaciones. Con el objetivo de contribuir a esta estrategia, se creó un proyecto de colaboración entre esta entidad y el Centro de Gobierno Electrónico adscrito a la Universidad de las Ciencias Informáticas, orientado hacia la informatización de los procesos que se llevan a cabo en los tribunales cubanos. De este modo surge el Expediente Judicial Electrónico como proyecto, el cual contempla la informatización del proceso de creación de los expedientes. Estos constituyen documentos que contienen un conjunto ordenado de antecedentes sobre una determinada cuestión en las diferentes instancias. Estos pasan por diferentes estados durante el proceso judicial, de los cuales el personal jurídico que interviene en ese expediente debe ser notificado. Es por ello que la presente investigación tiene como objetivo desarrollar los componentes para la gestión de alertas y notificaciones en dicha herramienta, de forma tal que se contribuya a la celeridad de estos procesos en los tribunales cubanos. Para lograr este objetivo, se utilizó como metodología de desarrollo el Proceso Unificado Ágil, versión establecida por la Universidad de las Ciencias Informáticas, así como un conjunto de tecnologías definidas por el equipo de arquitectura del proyecto, permitiendo de esta forma el desarrollo de la presente investigación. Los resultados fueron validados a través de métricas y pruebas de software, donde se pudo comprobar la calidad de los artefactos, así como de las funcionalidades internas y externas de los componentes propuestos.

Palabras claves: alertas judiciales, celeridad, expedientes judiciales electrónicos, notificaciones judiciales.

ÍNDICE DE CONTENIDOS

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

Los Tribunales Populares Cubanos (TPC) componen un sistema de órganos estatales, estructurados con independencia funcional de cualquier otro, y solo subordinados jerárquicamente a la Asamblea Nacional del Poder Popular y al Consejo de Estado. Para la correcta administración de justicia, los TPC se encuentran estructurados en tres instancias: Tribunal Supremo Popular (TSP), Tribunales Provinciales Populares (TPP) y Tribunales Municipales Populares (TMP). En estas instancias se llevan a cabo diferentes procesos distribuidos en cinco materias: Penal, Administrativo, Civil, Laboral y Económico.

Los TPC han dado pasos de avance en el uso de las tecnologías debido a que el país se ha propuesto desarrollar soluciones internas a problemas de gran impacto en la sociedad, es aquí donde juega un papel fundamental la Universidad de las Ciencias Informáticas (UCI), la cual está organizada por centros productivos con fines específicos. Adscrito a ella se encuentra el Centro de Gobierno Electrónico (CEGEL), el cual surge con el objetivo de informatizar procesos en las instituciones gubernamentales del país y con ello la posibilidad de agilizar los canales de información y ahorrar tiempo y recursos en las acciones que se realizan en dichas organizaciones.

Con el objetivo de contribuir a esta estrategia, se creó un proyecto de colaboración entre el TSP y el CEGEL. De este modo surge el Expediente Judicial Electrónico (XEJEL) como proyecto, el cual contempla la informatización del proceso de creación de los expedientes. Estos constituyen documentos que contienen un conjunto ordenado de antecedentes sobre una determinada cuestión en las diferentes instancias. Esta es una de las actividades en la cual se utiliza mayor tiempo y recurso. Los expedientes en los TPC pasan por diferentes estados durante el proceso judicial, de los cuales el personal jurídico que interviene en ese expediente debe ser notificado.

Estas notificaciones se realizan a los fiscales, a los representantes de las partes, así como a los acusados o sus defensores sobre las sentencias definitivas. De igual forma se notifica cuando uno de los involucrados en el caso tramita un expediente y una vez cumplido el tiempo o estado definido para un trámite determinado mediante Diligencias[1] y los representantes de las partes sobre los Autos que resuelven incidentes. Además, los secretarios dan cuenta al Tribunal con los escritos que le presenten o reciban y, sin la menor demora y bajo su responsabilidad, tienen la obligación de poner en conocimiento del Tribunal el vencimiento de los términos judiciales, consignándolo así por medio de Diligencias. Todos estos procesos se realizan de forma manual, lo que trae consigo que se invierta tiempo, recursos humanos y materiales.

Por otro lado, aunque forma parte de las funciones de los secretarios, no se cuenta con una manera de alertar constantemente al personal jurídico sobre el vencimiento de los términos judiciales. Esto

[1] Son los documentos extendidos por el secretario judicial con objeto de dejar constancia en los autos de determinados actos procesales o de interés procesal para el pleito correspondiente (Juridica, 2014).

provoca que en ocasiones los trámites a realizar sobre dichos expedientes estén fuera del término reglamentado por la Ley del Procedimiento Penal, para el caso de la materia Penal, y la Ley del Procedimiento Civil, Administrativo, Laboral y Económica, para el resto de las materias. De no existir estas dificultades los procesos se realizarían con mayor agilidad y permitiría al personal jurídico estar al tanto del estado de los trámites a realizar sobre los expedientes.

Ante la problemática anteriormente planteada surge el siguiente **problema a resolver:** ¿Cómo gestionar las alertas y notificaciones en el Expediente Judicial Electrónico de forma tal que contribuya a la celeridad de sus procesos?

Tomando en cuenta el problema antes propuesto se define como **objeto de estudio:** proceso de desarrollo de expedientes judiciales electrónicos.
De esta forma se determina como **objetivo general:** desarrollar los componentes para la gestión de alertas y notificaciones en la herramienta informática Expediente Judicial Electrónico de forma tal que contribuya a la celeridad de sus procesos.
Para ello se identifica como **campo de acción:** la gestión de alertas y notificaciones en el Expediente Judicial Electrónico.

Teniendo como referencia el problema a resolver y el objetivo general se plantea la siguiente **idea a defender**: con el desarrollo de los componentes para la gestión de alertas y notificaciones en la herramienta informática Expediente Judicial Electrónico, se contribuirá a la celeridad de sus procesos.

Una vez finalizado el presente trabajo, se tendrá como **resultado** los componentes para la gestión de alertas y notificaciones en la herramienta informática Expediente Judicial Electrónico.

Objetivos específicos:

- Elaborar el marco teórico de la investigación mediante el estudio de los referentes teóricos sobre el desarrollo de expedientes judiciales electrónicos.
- Realizar la especificación de los requisitos, así como el análisis y diseño de los componentes para la gestión de alertas y notificaciones del Expediente Judicial Electrónico como aproximación a la implementación.
- Implementar los componentes para la gestión de alertas y notificaciones del Expediente Judicial Electrónico para contribuir a la celeridad sus procesos.
- Validar el correcto funcionamiento de los componentes aplicando métricas y pruebas de software para avalar la calidad de los mismos.
- Validar las variables de la investigación para comprobar cómo la propuesta de solución contribuye a la celeridad en la gestión de alertas y notificaciones en el XEJEL.

Para dar cumplimiento a los objetivos propuestos se definen las siguientes **tareas de la investigación**:

1. Definición de los principales conceptos relacionado con el objeto de estudio y campo de acción.
2. Descripción del proceso de desarrollo de software y su metodología.
3. Caracterización de las tecnologías y las herramientas a utilizar para el desarrollo de la solución.
4. Especificación de los requisitos funcionales y no funcionales de la solución.
5. Descripción de la arquitectura de software base para la implementación del sistema.
6. Caracterización y selección de los patrones de diseño más factibles para la propuesta de solución.
7. Realización del diagrama de clases de diseño, modelo de datos y diseño de casos de pruebas.
8. Implementación de la propuesta de solución.
9. Ejecución de las pruebas de software para evaluar la calidad de la implementación.
10. Solución de las no conformidades al finalizar cada iteración de las pruebas.
11. Validación de las variables de la investigación.

Para dar solución a las tareas antes propuestas se definen los métodos científicos, clasificados en teóricos y empíricos. De los cuales se emplearon:

Métodos teóricos:

- **Histórico-lógico:** se empleó para el estudio de la trayectoria de la evolución de la herramienta informática Expediente Judicial Electrónico y así obtener un conocimiento previo sobre las leyes generales del funcionamiento y desarrollo en el proceder jurídico en los TPC para el caso de las alertas y notificaciones.
- **Analítico-sintético:** se empleó para el análisis del negocio en general, dividiéndolo en pequeñas partes para facilitar su estudio, y para determinar los elementos más importantes del negocio.
- **Modelación:** se utilizó para la realización de los diagramas necesarios en el proceso de desarrollo de software, haciendo una representación abstracta de la solución, facilitando así el desarrollo de la misma.

Métodos empíricos:

- **Entrevista:** permitió el intercambio verbal con el cliente para obtener la mayor cantidad de información posible, entender todo el proceso de negocio, comprender la estructura y funcionamiento de la organización, definir las deficiencias existentes que permitieron definir el problema a resolver y establecer el objeto de estudio.

- **Encuesta:** permitió mediante un cuestionario pre-elaborado, obtener información sobre el tiempo real, recursos humanos y materiales que utilizan los especialistas funcionales jurídicos para notificar y alertar a los involucrados en un expediente al estar sujeto a un trámite dentro de lo reglamentado por la ley.

El presente trabajo de diploma está estructurado en 3 capítulos de la siguiente forma:

Capítulo 1: Fundamentación teórica:
En este capítulo se definirán los conceptos asociados con el objeto de estudio y campo de acción, con el objetivo de lograr un mayor entendimiento del problema a resolver. De igual forma se realizará un estudio de sistemas similares a nivel nacional e internacional, de forma tal que sirva como punto de partida para la solución propuesta. Además, basándose en las orientaciones de la infraestructura de producción de la UCI, se caracterizará la metodología de desarrollo que guiará la presente investigación. Finalmente, se caracterizarán las herramientas y tecnologías utilizadas, las cuales fueron previamente definidas por el equipo de arquitectura del proyecto al que pertenece la solución propuesta.

Capítulo 2: Descripción de la solución propuesta:
En este capítulo se realizará una descripción de la solución propuesta a partir de las fases de la metodología definida para guiar el desarrollo de esta. Además, se llevará a cabo la disciplina de Requisitos para definir las funcionalidades que deben tener los componentes propuestos. Se describirá de igual forma la disciplina de análisis y diseño a través de los patrones de diseño, los diagramas de clases, el modelo de datos y el patrón arquitectónico que da soporte a la propuesta de solución. Por último, se enunciarán los estándares de codificación establecidos por parte del equipo de arquitectura y que deben tenerse en cuenta durante la implementación de la solución propuesta.

Capítulo 3: Validación de la solución propuesta:
En este capítulo se definirá la validación del diseño utilizando las métricas y haciendo un análisis de sus resultados. De igual forma, siguiendo la disciplina de Pruebas internas que propone la metodología AUP-UCI, se verificará el resultado de la implementación desarrollándose pruebas funcionales, unitarias y de rendimiento, esta última para fundamentar cómo la solución propuesta contribuye a la celeridad en el proceso de alertas y notificaciones en los TPC. Luego se procederá a la disciplina de Pruebas de liberación para avalar el correcto funcionamiento de los componentes de alertas y notificaciones del XEJEL antes de ser entregados al cliente.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

En el presente capítulo se definen los conceptos asociados con el objeto de estudio y campo de acción, con el objetivo de lograr un mayor entendimiento del problema a resolver. De igual forma se realiza un estudio de sistemas similares a nivel nacional e internacional, de forma tal que se utilice como punto de partida para la solución propuesta. Además, basado en las orientaciones de la infraestructura de producción de la UCI, se caracteriza la metodología de desarrollo que guía la presente investigación. Por último, se caracterizan las herramientas y tecnologías utilizadas, las cuales fueron previamente definidas por el equipo de arquitectura del proyecto al que pertenece la solución propuesta.

1.1 Definición de los principales conceptos relacionado con la investigación

Con el objetivo de lograr un entendimiento del contenido a tratar en la investigación, se requiere el conocimiento de los conceptos enunciados a continuación:

Expediente judicial: es el conjunto de documentos electrónicos correspondientes a un procedimiento judicial, cualquiera que sea el tipo de información que contenga. Es propiedad del proceso mismo por eso debe seguir paso a paso el desarrollo de la disensión, la participación de las partes, sus propuestas y desacuerdos, las pruebas necesarias, etc. El expediente judicial finaliza con la sentencia en Primera Instancia, sin embargo, si esta es apelada, debe continuar con la intervención de la cámara, los actos de la segunda instancia y su resolución (MJU, 2016).

Alerta: el término alerta, del italiano *allerta*, hace referencia a una situación de vigilancia o atención. Un estado o una señal de alerta es un aviso para que se extremen las precauciones o se incremente la vigilancia (MJU, 2016).

Notificación: el término 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). Una notificación judicial, por otra parte, es un acto de comunicación de un juzgado o tribunal. Este documento debe ser entregado a la persona o ser publicado a través de un edicto[2] para que el destinatario conozca el lugar, la fecha y la hora en que debe presentarse a prestar declaración o intervenir por una causa judicial (Gardey, 2019).

Celeridad: el concepto de celeridad deriva del vocablo latino *celerĭtas*. Se trata de un término que hace referencia a la velocidad, la premura, la rapidez o la prisa (Gardey, 2019). En la presente investigación se adopta este concepto vinculado con el tiempo que se demora el personal jurídico al realizar una

[2] Edicto es el mandato o decreto publicado con autoridad de un tribunal.

actividad sobre un expediente determinado, en este caso, al realizar las alertas sobre el vencimiento del término de las actuaciones o del estado por el cual se transita.

Una vez definidos los conceptos fundamentales que engloban el objeto de estudio y el campo de acción, a continuación, se muestra el estudio de soluciones similares a nivel nacional e internacional.

1.2 Estudio de soluciones similares

En la actualidad se promueve el uso de la informática en la rama del derecho, y aunque su avance es lento, existen numerosas soluciones informáticas en el mundo que hacen uso de sistemas informáticos para la gestión procesal, a continuación, se muestra una breve descripción de las estudiadas en la presente investigación.

1.2.1 Sistemas estudiados a nivel internacional

Sistema de gestión de notificaciones telemáticas (Lexnet): se basa en un sistema de correo electrónico seguro, con firma electrónica, a través del cual el usuario recibe las notificaciones emitidas por el juzgado y presenta los escritos por vía telemática. Posteriormente, y a través del mismo sistema, recibe un resguardo electrónico en el que se acredita que la transmisión se ha efectuado correctamente y se le comunica la fecha efectiva de la presentación de dicho escrito en la oficina judicial o juzgado correspondiente. Su uso se regula en el Real Decreto 84/2007, de 26 de enero, sobre implantación en la administración de justicia del sistema informático de telecomunicaciones Lexnet para la presentación de escritos y documentos, el traslado de copias y la realización de actos de comunicación procesal por medios telemáticos. Dentro del marco funcional que comprende la realización de actos de comunicación, la presentación de escritos con traslado de copias y de escritos iniciadores, el sistema Lexnet posee las siguientes funcionalidades: presentación de documentos y adjuntos al órgano judicial, envío de notificaciones por parte del órgano judicial, emisión de acuses de recibo, acceso autenticado y seguro mediante navegador web, acceso e intercambio de documentos mediante servicios web, gestión de carpetas de usuario, búsqueda y traza de mensajes (IT-CGAE, 2013).
Este sistema se basa solo en la gestión de notificaciones y documentos, funcionalidades necesarias en la solución a implementar, pero que por sí solas no resuelven los problemas de los TPC, sin embargo, aportan ideas a la solución como el intercambio de información mediante servicios web.

Sistema para la gestión jurídica (Lex-Doctor): este completo sistema es utilizado en la amplia mayoría de los estudios jurídicos y asesorías letradas informatizados de Argentina. Desarrollado por Sistemas Jurídicos SRL, empresa que tiene un importante posicionamiento en el segmento de los sistemas aplicados a la actividad jurídica en Latinoamérica. Su tecnología de administración de datos,

permite manejar grandes volúmenes de información, y organizar grupos de trabajo operando en redes de gran cantidad de terminales. Entre sus principales funcionalidades se encuentran:

- Gestión de expedientes: Procesos judiciales, extrajudiciales, mediaciones y todo tipo de expediente de índole jurídico. Partes, letrados, agendas, pruebas, gestiones, movimientos, audiencias, vencimientos. Manejo total de la procuración.
- Gestión económica: Cuentas por cliente, por expediente, y otros tipos de cuenta definibles por el usuario. Actualizaciones y liquidaciones, con conversión de monedas. Liquidaciones individuales o masivas.
- Gestión documental: Confección automatizada de escritos, cédulas, oficios, mandamientos, telegramas, cartas, y otros documentos, con posibilidad de confección seriada. Almacenamiento de documentos creados por el sistema y por otros programas instalados en la PC.
- Reportes y estadísticas: Confección de listados e informes, con diseños programados o propios, y con posibilidad de exportar a distintos formatos. Evaluación estadística gráfica.
- Acceso remoto: Opcionalmente, permite operar a través de Internet; así, disponiendo de una conexión y una PC, se puede trabajar en línea con la oficina desde cualquier lugar (SRL, 2011).

Lex-Doctor a pesar de ser un sistema con mucho prestigio y aceptación presenta diferentes inconvenientes para su utilización por parte de los TPC, puesto que el mismo está desarrollado sobre software propietario, impidiendo las posibilidades de modificación y adaptación al medio nacional, además centra su trabajo sobre la gestión documental fundamentalmente, y para su utilización es necesario comprar la licencia de uso.

Expediente Judicial Electrónico en España (EJE): el expediente judicial electrónico permite que todos los intervinientes judiciales accedan al mismo expediente y documentación, lo que ahorra tiempo, optimiza recursos y agiliza la tramitación de los procedimientos. El proceso de implantación del expediente judicial electrónico precisa de un trabajo previo de digitalización de todos los documentos de los expedientes, bajo un sistema seguro y de manera certificada, así como la implantación de un sistema de gestión documental. La digitalización de los expedientes también mejora la accesibilidad a la información por parte de los profesionales judiciales autorizados, que podrán acceder a un mismo documento desde distintos canales y al mismo tiempo. Además, también favorece la comunicación entre órganos jurisdiccionales (MJU, 2016).

Este sistema, a pesar de tener grandes ventajas, no es compatible con el sistema judicial cubano, además de tener un proceso de digitalización, el cual no es objetivo de los TPC en un futuro inmediato.

1.2.2 Sistemas estudiados a nivel nacional

Sistema para la tramitación de procesos penales (SisProp): sistema informático desarrollado en la provincia de Villa Clara en el 2008. La propuesta inicial fue que abarcara la instancia suprema y

provincial de la materia penal, desarrollándose solamente la tramitación de los procesos en la última instancia. Facilita la tramitación de los procesos penales, con agilidad y precisión, en las entidades del sistema de tribunales populares del país. Dos de las características fundamentales del sistema son su seguridad dada la naturaleza de la información que se maneja, y la flexibilidad con respecto a cualquier modificación que pudiera sufrir la Ley Procesal Penal. Este software presenta algunas deficiencias como son:

- No capta ningún dato de la fase judicial de la tramitación y decisión del tribunal.
- No aporta estadística, ni información alguna.
- No posee una exhaustiva validación de los datos.
- No se trabajó con un NIP[3] de cada proceso.
- Programado en Delphi, corre sobre SQL Server por lo que no es compatible con el software libre (Castro Morell, y otros, 2008).

Este sistema no cumple con las expectativas para las que fue creado, no es posible integrarlo a la herramienta informática Expediente Judicial Electrónico debido a que las tecnologías de desarrollo no son compatibles con las definidas para dicha herramienta, ya que estas son más avanzadas.

Sistema para la tramitación de los procesos en la materia Económica (SisEco): sistema informático desarrollado en La Habana en el 2002, concebido para el área de estadística en el procedimiento Económico. El sistema cuenta con funcionalidades muy básicas y es lento en cuanto a tiempo de respuesta. Como principales deficiencias se tiene:

- Inserta documentos radicados manualmente. La secretaria de estadística recoge el Libro de Radicación de Escritos (LRE) e inserta los datos en la aplicación y cada cinco años borra los datos de los años anteriores quedando solamente la información asentada en los libros.
- Las salvas se guardan en disquetes (Juiz, 2009).

Al igual que la anterior, esta aplicación no cumplió las expectativas iniciales de su creación. Es lento, característica que no va aparejada con la estadística en tiempo real. Por otra parte, informatiza solo una pequeña parte del proceso, pues la radicación se realiza de forma manual, las salvas se guardan en disquetes técnica que ha quedado obsoleta en nuestros días, la información almacenada es borrada cada cinco años, algo incomprensible para un sistema judicial para el cual mantener registros es primordial, ya que la información es el mecanismo principal para la justicia. Teniendo en cuenta lo antes explicado se lleva a la conclusión de que tampoco es posible integrarlo a la herramienta informática Expediente Judicial Electrónico.

[3] Número de Identificación Permanente o General.

Sistema de Información para la Gestión de los Tribunales Populares Cubanos (SITPC): este sistema, basado en tecnología web, posibilita entre otras funcionalidades generales, la presentación digital de escritos, la radicación automática de expedientes, el procesamiento de textos con modelos de documentos, y las notificaciones electrónicas a las partes; todo desde la perspectiva del expediente digital (González Ochoa, y otros, 2018).

SITPC en su totalidad está compuesto por 46 módulos, de los cuales se encuentran implementados 7, de la materia Administrativa, el Administrativo, de la materia Económico, el Ejecutivo, de la materia Penal, el Ordinario de la instancia provincial y municipal, de la materia Civil, Actos preparatorios y el Ordinario y de la materia Laboral, Disciplina y Derecho Laboral, por lo que el resto de los módulos que responden a los procesos judiciales que se atienden en los TPC aún se llevan a cabo de forma manual. Por esta razón se concluye que no es posible integrar la funcionalidad de notificaciones electrónicas a la herramienta Expediente Judicial Electrónico.

1.3 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 (Enríquez Ruiz, y otros, 2017).

Como metodología de desarrollo de software, para el desarrollo de la solución propuesta, se decide seleccionar la definida por la UCI. Esta es una variación del Proceso Unificado Ágil (AUP), ya que se adapta al ciclo de vida productivo de dicha universidad. A continuación, se describen los principales elementos de la misma:

1.3.1 Proceso Unificado Ágil variación UCI

Al no existir una metodología de software universal, ya que toda metodología debe ser adaptada a las características de cada proyecto, exigiéndose así que el proceso sea configurable, se decide, por parte de la UCI, hacer una variación de la metodología AUP, de forma tal que se adapte al ciclo de vida definido para la actividad productiva de la UCI. Una metodología de desarrollo de software tiene entre sus objetivos aumentar la calidad del software que se produce, de ahí la importancia de aplicar buenas prácticas, para ello nos apoyaremos en el Modelo CMMI-DEV v1.3[4] (Rodríguez Sánchez, 2015).

Con la adaptación de AUP que se propone para la actividad productiva de la UCI:

- Se logra estandarizar el proceso de desarrollo de software, dando cumplimiento además a las buenas prácticas que define CMMI-DEV v1.3.

[4] Modelo de Madurez de Capacidades Integrado para Desarrollo (CMMI-DEV) proporciona buenas prácticas para el desarrollo y mantenimiento de productos y servicios.

- Se logra hablar un lenguaje común en cuanto a fases, disciplinas, roles y productos de trabajos (Rodríguez Sánchez, 2015).

AUP propone 7 disciplinas (Modelo, Implementación, Prueba, Despliegue, Gestión de configuración, Gestión de proyecto y Entorno), se decide para el ciclo de vida de los proyectos de la UCI tener 7 disciplinas también, pero a un nivel más atómico que el definido en AUP: Modelado de negocio, Requisitos, Análisis y diseño, Implementación, Pruebas Internas, Pruebas de Liberación y Pruebas de Aceptación (Rodríguez Sánchez, 2015).
De igual forma se definen por parte de la metodología 4 escenarios para la disciplina de Requisitos. Para el caso del presente trabajo se decide utilizar el Escenario 4 para la encapsulación de los requisitos de los componentes propuestos, a través de las Historias de Usuario. Este se tuvo en cuenta ya que es el seleccionado por parte del equipo de XEJEL. Además, al evaluar el negocio a informatizar se obtuvo como resultado un negocio bien definido. Por otra parte, el cliente estará siempre acompañando al equipo de desarrollo para convenir los detalles de los requisitos y así poder implementarlos, probarlos y validarlos. Por último, se recomienda en proyectos no muy extensos, ya que una historia de usuario no debe poseer demasiada información.

1.4 Caracterización de las herramientas y lenguajes a utilizar

Un lenguaje de programación es un lenguaje formal diseñado para realizar procesos que pueden ser llevados a cabo por máquinas como las computadoras (Olarte Gervacio, 2018). Por otra parte, las herramientas informáticas, son programas, aplicaciones o simplemente instrucciones usadas para efectuar otras tareas de modo más sencillo (Sistema-Master-Magazine, 2018).
En este epígrafe se describen las herramientas y lenguajes a utilizar definidos por el equipo de arquitectura del proyecto de XEJEL, para el desarrollo de la propuesta de solución.

1.4.1 Herramientas CASE[5]

Las herramientas CASE son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas 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 costes, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores entre otras. Proporcionan ayuda automatizada a las actividades del proceso de desarrollo de software (Menéndez-Barzanallana Asensi, 2016). A continuación, se describe la herramienta CASE que ha sido

[5] Ingeniería de Software Asistida por Computadoras, CASE por sus siglas en inglés de Computer Aided Software Engineering

concebida para soportar el ciclo de vida completo del proceso de desarrollo del software a través de la representación de todo tipo de diagramas.

Visual Paradigm

Visual Paradigm propicia un conjunto de ayudas para el desarrollo de programas informáticos, desde la planificación, pasando por el análisis y el diseño, hasta la generación del código fuente de los programas y la documentación. Constituye una herramienta privada disponible en varias ediciones, cada una destinada a satisfacer diferentes necesidades: (Enterprise, Professional, Community, Standard, Modeler y Personal)[6]. Fue diseñado para una amplia gama de usuarios interesados en la construcción de sistemas de software de forma fiable a través de la utilización de un enfoque Orientado a Objetos. Se caracteriza por:

- Disponibilidad en múltiples plataformas (Windows, Linux).
- Diseño centrado en casos de uso y enfocado al negocio que generan un software de mayor calidad.
- Uso de un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación.
- Modelo y código que permanece sincronizado en todo el ciclo de desarrollo.
- Soporta aplicaciones Web.
- Varios idiomas.
- Fácil de instalar y actualizar.
- Compatibilidad entre ediciones.
- Capacidades de ingeniería directa e inversa.
- Generador de informes (Paradigm, 2017).

Para el desarrollo del XEJEL se decide, por parte del equipo de arquitectura, utilizar Visual Paradigm for UML en su versión 8.0, debido a que la UCI posee una licencia académica para su uso. Además, por la fácil integración con el lenguaje PHP y a la experiencia del equipo de desarrollo en su empleo, lo que posibilita reducir el tiempo en el proceso de desarrollo del software, ya que no hay necesidad de capacitar a los desarrolladores.

1.4.2 Lenguaje de modelado UML v2.0

El lenguaje UML es un estándar OMG[7] diseñado para visualizar, especificar, construir y documentar software orientado a objetos. Un modelo es una simplificación de la realidad. El modelado es esencial en la construcción de software para:

- Comunicar la estructura de un sistema complejo.
- Especificar el comportamiento deseado del sistema.

[6] Empresa, Profesional, Comunidad, Estándar, Modelador y Personal.
[7] Grupo de administración de objetos (**OMG** por sus siglas en inglés de: *Object Management Group*)

- Comprender mejor lo que estamos construyendo.
- Descubrir oportunidades de simplificación y reutilización.

Un modelo proporciona “los planos” de un sistema y puede ser más o menos detallado, en función de los elementos que sean relevantes en cada momento (Larman, 2016).

Para el desarrollo del XEJEL se decide, por parte del equipo de arquitectura, utilizar UML en su versión 2.0 y, por consiguiente, para el desarrollo de la solución propuesta.

1.4.3 Symfony

Symfony es un completo marco de trabajo diseñado para optimizar, gracias a sus características, el desarrollo de las aplicaciones web. Para empezar, separa la lógica de negocio, la lógica de servidor y la presentación de la aplicación web. Proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación web compleja. Además, automatiza las tareas más comunes, permitiendo al desarrollador dedicarse por completo a los aspectos específicos de cada aplicación (Potencier, et al., 2008).

Para el desarrollo del XEJEL se utiliza Symfony en su versión 3.4, este posee la facilidad de gestionar las rutas URL [8] inteligentes, permitiendo configurar direcciones amigables en las páginas de la aplicación. Permite su integración con las bibliotecas de otros fabricantes y así aprovechar las ventajas en la parte del *frontend* de nuevas librerías tanto en JavaScript tipo Angular, o jQuery, entre otras. Por último, es la versión LTS[9] más actual, lo que facilita incorporar cada uno de estos elementos en el XEJEL.

1.4.4 Bootstrap

Es un marco de trabajo de software libre para diseño de sitios y aplicaciones web. Contiene plantillas de diseño con tipografía, formularios, botones, cuadros, menús de navegación y otros elementos de diseño basado en HTML y CSS, así como, extensiones de JavaScript opcionales. Los desarrolladores eligen en un formulario los componentes y ajustes deseados, y de ser necesario, los valores de varias opciones a sus necesidades. El paquete consecuentemente generado ya incluye la hoja de estilo pre-compilada (Getbootstrap.com, 2018).

Se decide la utilización de Bootstrap en su versión 4.1 debido a las nuevas funcionalidades que el mismo integra, brinda la posibilidad de asignarle una altura y ancho a los elementos de la página web de forma automática por lo que se adaptará a todos los dispositivos. Además, provee una galería de iconos de forma gratuita e incorpora el tratamiento de los *tooltips*, de forma tal que, al colocar el puntero sobre un elemento, este mostrará de una manera diferente el texto o título correspondiente.

[8] Localizador Uniforme de Recursos (**URL** por sus siglas en ingles de: *Uniform Resource Locator*)
[9] Soporte a largo plazo (**LTS**, por sus siglas en inglés de: *Long Term Support*)

1.4.5 Angular

Angular es una plataforma y un marco para la creación de aplicaciones cliente en HTML y TypeScript. Angular está escrito en TypeScript. Implementa la funcionalidad central y opcional como un conjunto de bibliotecas de TypeScript que importa a sus aplicaciones. Los bloques de construcción básicos de una aplicación Angular son NgModules, que proporcionan un contexto de compilación para los componentes. NgModules recopila códigos relacionados en conjuntos funcionales; una aplicación Angular se define por un conjunto de NgModules. Una aplicación siempre tiene al menos un módulo raíz que habilita el arranque, y normalmente tiene muchos más módulos de características:

- Los componentes definen vistas, que son conjuntos de elementos de pantalla que Angular puede elegir y modificar según la lógica y los datos de su programa.
- Los componentes utilizan servicios, que proporcionan una funcionalidad específica que no está directamente relacionada con las vistas. Los proveedores de servicios se pueden inyectar en los componentes como dependencias, haciendo que su código sea modular, reutilizable y eficiente.

Tanto los componentes como los servicios son simples clases, con decoradores que marcan su tipo y proporcionan metadatos que le indican a Angular cómo usarlos. Los metadatos de una clase de componente lo asocian con una plantilla que define una vista. Una plantilla combina HTML ordinario con directivas angulares y un marcado de enlace que permite a Angular modificar el HTML antes de representarlo. Los metadatos para una clase de servicio proporcionan la información que Angular necesita para que esté disponible para los componentes a través de la inyección de dependencia. Los componentes de una aplicación típicamente definen muchas vistas, ordenadas jerárquicamente. Angular proporciona el servicio de enrutador para ayudarlo a definir las rutas de navegación entre las vistas. El enrutador proporciona sofisticadas capacidades de navegación en el navegador (Jiménez Castela, 2017).

Para el desarrollo del XEJEL se decide, por parte del equipo de arquitectura, utilizar Angular en su versión 6.0 y, por tanto, para el desarrollo de la solución propuesta.

1.4.6 Lenguajes de programación

Un lenguaje de programación es aquella estructura que, con una cierta base sintáctica y semántica, imparte distintas instrucciones a un programa de computadora. Existen diversos lenguajes de programación, lo que ha llevado al desarrollo de intérpretes[10] y compiladores[11] (Merino, 2012).

A continuación, se describen cada uno de los lenguajes utilizados por el marco de trabajo definido y, por consiguiente, en el desarrollo de la propuesta de solución.

HTML

[10] Programas que adaptan las instrucciones encontradas en otro lenguaje (Gardey, 2019).
[11] Aquellos programas que traducen de un lenguaje a otro (Gardey, 2019).

HTML es un lenguaje de marcado que se utiliza para el desarrollo de páginas de Internet. Se trata de la sigla que corresponde a *HyperText Markup Language*, es decir, Lenguaje de Marcas de Hipertexto, que podría ser traducido como Lenguaje de Formato de Documentos para Hipertexto. HTML en su versión 5 establece una serie de nuevos elementos y atributos que reflejan el uso típico de los sitios web modernos. Mejora el elemento <canvas>[12], capaz de renderizar elementos 3D en los navegadores, por ejemplo: Firefox, Chrome y Opera (W3C, 2018).

En el caso de XEJEL se decide la utilización de HTML en su versión 5, ya que es la versión seleccionada por el equipo de arquitectura del proyecto al que pertenece la propuesta de solución.

CSS

CSS es un lenguaje de marcado que se emplea para dar formato a un sitio web. Es decir, funciona en conjunto con los archivos HTML. Por esta razón, para crear un sitio web se debe saber tanto HTML como CSS. Cabe agregar que el lenguaje CSS3 se puede aplicar en la misma hoja en la que se está desarrollando un documento HTML, pero por motivos de productividad se suele realizar en un documento aparte con la extensión css. Este documento se puede vincular a cada página HTML que conforme el sitio web, es por ello que es más útil realizar los estilos por separado. CSS3 funciona mediante módulos, algunos de los más comunes son "*colors*", "*fonts*", "*backgrounds*", etc. Los módulos son solo categorías en las que se pueden dividir las modificaciones que hacemos al aspecto de nuestro sitio web (Guevara, 2017).

Se decide la utilización de CSS en su versión 3, ya que es la versión seleccionada por el equipo de arquitectura del proyecto al que pertenece la propuesta de solución.

JavaScript

JavaScript es un lenguaje de programación que se utiliza principalmente para crear páginas web dinámicas. Una página web dinámica es aquella que incorpora efectos como texto que aparece y desaparece, animaciones, acciones que se activan al pulsar botones y ventanas con mensajes de aviso al usuario. Técnicamente, JavaScript es un lenguaje de programación interpretado, por lo que no es necesario compilar los programas para ejecutarlos. En otras palabras, los programas escritos con JavaScript se pueden probar directamente en cualquier navegador sin necesidad de procesos intermedios.

- Es liviano.
- Multiplataforma, ya que se puede utilizar en Windows, Linux o Mac o en el navegador que desee el usuario.

[12] <canvas> es un elemento HTML el cual puede ser usado para dibujar gráficos usando scripts.

- Es imperativo y estructurado, mediante un conjunto de instrucciones indica al computador qué tarea debe realizar.
- Prototipado, debido a que usa prototipos en vez de clases para el uso de herencia.
- Orientado a objetos y eventos.
- Es Interpretado, no se compila para poder ejecutarse (Grados Caballero, 2018).

Se utiliza JavaScript en su versión 1.8, ya que es la seleccionada por el equipo de arquitectura del proyecto al que pertenece la propuesta de solución.

TypeScript

TypeScript es un lenguaje de programación de distribución y código abierto desarrollado por Microsoft. Su propio fabricante lo define como un *superset* (superconjunto) de JavaScript que, en esencia, mejora JavaScript añadiéndole funciones de tipado y de programación orientada a objetos. Además, la sintaxis y desarrollo en TypeScript incorpora las mejoras y características de ECMAScript 6, estandarización de JavaScript. Debido a estas nuevas especificaciones, el framework Angular utiliza TypeScript como lenguaje de programación para la lógica de las aplicaciones (Jiménez Castela, 2017).

Se decide la utilización de TypeScript en su versión 3.2, ya que es la requerida para la versión de angular seleccionada por el equipo de arquitectura del proyecto al que pertenece la propuesta de solución.

PHP

PHP [13] es un lenguaje interpretado en el lado del servidor que se caracteriza por su potencia, versatilidad, robustez y modularidad. Los programas escritos en PHP son embebidos directamente en el código HTML y ejecutados por el servidor Web a través de un intérprete antes de transferir al cliente que lo ha solicitado un resultado en forma de código HTML puro. Al ser un lenguaje que sigue las corrientes de código abierto, son totalmente accesibles de forma gratuita en la red. Es un lenguaje multiplataforma, que puede funcionar sobre la mayoría de los servidores Web y brinda soporte a más de 20 tipos de base de datos.

Su rapidez en la ejecución y los bajos requisitos de consumo en los sistemas donde es desplegado lo hacen uno de los preferidos por los desarrolladores. Dispone de una conexión nativa a los principales sistemas de base de datos utilizados actualmente tales como Postgres, MySQL, Oracle, Microsoft SQL Server, lo cual permite la creación de aplicaciones web robustas. Su mayor ventaja radica en ser un lenguaje libre, por lo que se convierte en una alternativa de muy fácil acceso, además de contar con una comunidad de desarrolladores que intercambian experiencias lo que facilita la rápida solución de problemas sin costo alguno (Sæther Bakken, y otros, 2002).

[13] PHP: Hypertext Pre-processor

Para el desarrollo del XEJEL se decide, por parte del equipo de arquitectura, utilizar PHP en su versión 7.2 y, por tanto, para el desarrollo de la solución propuesta.

1.4.7 jQuery

jQuery a una librería o biblioteca de JavaScript que facilita la programación en este lenguaje. Por lo general se emplea para añadir elementos interactivos a una página web sin necesidad de tener que programar demasiado. En el terreno de la informática, las librerías o bibliotecas son colecciones de recursos e implementaciones que se encuentran codificadas en un determinado lenguaje de programación y que resultan funcionales. Un lenguaje de programación, en tanto, es un lenguaje formal que detalla instrucciones a una computadora (ordenador) para producir diferentes datos (Pérez Porto, 2018).

Para el desarrollo del XEJEL se decide, por parte del equipo de arquitectura, utilizar JQuery en su versión 3.3 y, por consiguiente, para el desarrollo de la solución propuesta.

1.4.8 Sistema gestor de bases de datos

Un Sistema gestor de base de datos es un sistema que permite la creación, gestión y administración de bases de datos, así como la elección y manejo de las estructuras necesarios para el almacenamiento y búsqueda de la información del modo más eficiente posible (Iruela, 2016).

PostgreSQL

Es un sistema de gestión de base de datos relacional orientada a objetos y libre, publicado bajo la licencia BSD[14].

Como muchos otros proyectos de código abierto, el desarrollo de PostgreSQL no es manejado por una empresa y/o persona, sino que es dirigido por una comunidad de desarrolladores que trabajan de forma desinteresada, altruista, libre y/o apoyada por organizaciones comerciales. La comunidad PostgreSQL se denomina el PGDG (PostgreSQL Global Development Group).

Sus principales características son:

- Alta concurrencia: mediante un sistema denominado MVCC (Acceso concurrente multiversión, por sus siglas en inglés).
- Amplia variedad de tipos nativos: provee nativamente varios soportes.
- Ahorros considerables de costos de operación.
- Estabilidad y confiabilidad (Iruela, 2016).

[14] Licencia de software otorgada principalmente para los sistemas BSD (Berkeley Software Distribution). Es una licencia de software libre permisiva. La licencia BSD permite el uso del código fuente en software no libre.

Para el desarrollo del XEJEL se decide, por parte del equipo de arquitectura, utilizar PostgreSQL en su versión 10.1 y, de igual forma, para el desarrollo de la solución propuesta.

1.4.9 PgAdmin

PgAdmin es un sistema completo de gestión y diseño de bases de datos PostgreSQL para sistemas Unix y Windows. Está disponible de forma gratuita según los términos de la licencia The PostgreSQL y puede redistribuirse siempre que se cumplan los términos de la licencia. El proyecto es administrado por el equipo de desarrollo pgAdmin. Este software fue escrito como un sucesor de los productos pgAdmin y pgAdmin II originales. Está escrito en C ++ y utiliza el excelente conjunto de herramientas multiplataforma wxWidgets (anteriormente wxWindows) (pgAdmin.org, 2016).

Para el desarrollo del XEJEL se decide, por parte del equipo de arquitectura, utilizar PgAdmin en su versión 1.22 y, por tanto, para la solución propuesta.

1.4.10 Herramienta de mapeo objeto relacional

El mapeo objeto relacional (ORM por sus siglas en inglés de *Object Relational Mapping*) es una técnica de programación para convertir datos del sistema de tipos utilizado en un lenguaje de programación orientado a objetos al utilizado en una base de datos relacional. En la práctica esto crea una base de datos virtual orientada a objetos sobre la base de datos relacional. Esto posibilita el uso de las características propias de la orientación a objetos, esencialmente la herencia y el polimorfismo (Enriquez, 2014).

Doctrine

Doctrine es un ORM escrito en PHP que proporciona una capa de persistencia para objetos PHP. Es una capa de abstracción que se sitúa justo encima de un sistema de gestión de bases de datos, además posee un código base muy estable y de alta calidad, así como extremadamente flexibles y potentes funciones de mapeo de objetos y consulta. De igual forma posee soporte para la programación de bases de datos de alto y bajo nivel, así como una gran comunidad e integraciones con muchos marcos de trabajo diferentes, como por ejemplo Symfony (Doctrine.org, 2018).

Para el desarrollo del XEJEL se decide, por parte del equipo de arquitectura, utilizar ORM Doctrine en su versión 2.4 y, de igual forma, para la propuesta de solución.

1.4.11 Servidor web

Un servidor Web es un programa que utiliza el protocolo de transferencia de hipertexto, HTTP (*Hypertext Transfer Protocol*), para servir los archivos que forman páginas Web a los usuarios, en respuesta a sus solicitudes, que son reenviados por los clientes HTTP de sus computadoras. Las computadoras y los dispositivos dedicados también pueden denominarse servidores Web (Rouse, 2019).

Apache

La aplicación Apache es un software de código abierto, una aplicación Java 100% pura diseñada para cargar el comportamiento funcional de la prueba y medir el rendimiento. Originalmente fue diseñado para probar aplicaciones web, pero desde entonces se ha expandido a otras funciones de prueba, rápida y eficiente, continuamente actualizado. Entre sus principales características están:

- Multiplataforma.
- Modular: Puede ser adaptado a diferentes entornos y necesidades, con los diferentes módulos de apoyo que proporciona, y con la API de programación de módulos, para el desarrollo de módulos específicos.
- Extensible: gracias a ser modular se han desarrollado diversas extensiones entre las que destaca PHP, un lenguaje de programación del lado del servidor (Apache.org, 2018).

Para el desarrollo del XEJEL se decide, por parte del equipo de arquitectura, utilizar Apache en su versión 2.4, ya que presenta mejoras en cuanto al rendimiento al minimizar el consumo de memoria y en las concurrencias de las peticiones, es la versión más rápida de Apache.

1.4.12 Entorno de desarrollo integrado

Entorno de desarrollo integrado (IDE por sus siglas en ingles de *Integrated Development Environment)* 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 (Martin Olivera, y otros, 2016).

NetBeans

NetBeans es un proyecto de código abierto dedicado a proporcionar productos de desarrollo de software sólidos (el IDE de NetBeans y la Plataforma de NetBeans) que atienden las necesidades de los desarrolladores, usuarios y las empresas que confían en NetBeans como base para sus productos; en particular, para permitirles desarrollar estos productos de forma rápida, eficiente y fácil al aprovechar las fortalezas de la plataforma Java y otros estándares relevantes de la industria (NetBeans.org, 2018). Se decide la utilización de NetBeans en su versión 8.2, ya que proporciona soporte para el cliente de control de versiones seleccionado por parte del equipo de arquitectura del proyecto XEJEL, lo que permite realizar tareas de control de versiones directamente desde el proyecto dentro del IDE.

1.4.13 Control de versiones

El control de versiones es un sistema que registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo, de modo que puedas recuperar versiones específicas más adelante (Murua, y otros, 2019).

Para el desarrollo del XEJEL, el equipo de arquitectura decide utilizar Git en su versión 2.17, teniendo en cuenta las siguientes características:

Git

Es un sistema de control de versiones gratuito, de código abierto, y se distribuye, o sea, todos los desarrolladores tienen el historial completo de su repositorio de códigos a nivel local. Esto hace que el clon inicial del repositorio sea más lento, pero las operaciones subsiguientes, como cometer, culpar, difuminar, fusionar y registrar mucho más rápido. Incluye las funcionalidades de crear ramas y merges[15], además de reescribir historiales de repositorios, lo cual ha dado como resultado muchas herramientas innovadoras y eficaces.

Básicamente, Git funciona del siguiente modo:

1. Crea un "repositorio" (proyecto) con una herramienta de alojamiento de Git
2. Copia (o clona) el repositorio a una máquina local
3. Añade un archivo al repositorio local y confirma ("*commit*") los cambios
4. Envía ("*push*") los cambios a la rama principal
5. Realiza cambios en un archivo con una herramienta de alojamiento de Git y se confirman
6. Extrae ("*pull*") los cambios a una máquina local
7. Crea una rama ("*branch*", versión), se realiza algún cambio y se confirma
8. Fusiona ("*merge*") a una rama con la rama principal (Murua, y otros, 2019).

1.5 Conclusiones del capítulo

El estudio de los conceptos asociados al problema facilitó una mejor comprensión en cuanto a la gestión de alertas y notificaciones en expedientes judiciales electrónicos. Luego del estudio de varios sistemas similares, se llegó a la conclusión que estos a pesar de poseer ventajas, no satisfacen las necesidades actuales del TSP para llevar a cabo el expediente judicial de forma electrónica. La caracterización de la metodología de desarrollo estandariza el ciclo de vida del software, dando cumplimiento además a las buenas prácticas que define la UCI. Por otra parte, el estudio de las características de las herramientas y lenguajes a utilizar fundamenta su selección por parte del equipo del XEJEL para el desarrollo de los componentes propuestos en la presente investigación.

[15] Hacer fusiones entre diferentes ramas

CAPÍTULO 2: DESCRIPCIÓN DE LA SOLUCIÓN PROPUESTA

En el presente capítulo se realiza una descripción general del posible resultado, luego se procede a realizar la disciplina de requisitos abarcando los requisitos funcionales y no funcionales. Por otra parte, se obtienen los artefactos como producto de la disciplina Análisis y diseño, como base a la implementación de la solución propuesta. Por último, se especifican los estándares de codificación concebidos por el equipo de arquitectura del XEJEL a tener en cuenta durante el desarrollo del resultado de la presente investigación.

2.1 Descripción general de la propuesta de solución

El resultado de la colaboración entre el TSP y el CEGEL ha dado paso al surgimiento de la herramienta informática XEJEL, con el propósito de agilizar del proceder con los expedientes que intervienen en los procesos. El empleo del expediente judicial en formato electrónico se materializa en la sustitución de su tradicional legajo por su equivalente en formato digital y, lo que es más importante, las herramientas y medidas necesarias para garantizar la correcta administración de justicia. Ejemplo de esto es la acción de notificar a los fiscales, a las personas involucradas en el proceso judicial, a los representantes de las partes, así como a los acusados o sus defensores sobre las sentencias definitivas, de igual forma cuando uno de los actores tramita un expediente y una vez cumplido el tiempo o estado definido para un trámite determinado. Por otra parte, una vez vencidos los términos judiciales el personal jurídico debe ser alertado, trayendo consigo que los expedientes estén dentro del término reglamentado por la ley. Para la configuración de las alertas dentro de los TPC, así sea en el propio tribunal o en alguna sala específica, se debe tener en cuenta la cantidad de días que un expediente se encuentre en un estado determinado. Para el caso de las configuraciones de las alertas en los tribunales, estas pueden tenerse en cuenta para algunas de las salas del propio tribunal.

2.2 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 Sánchez, 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, 2010).

Para la obtención o captura de requisitos en la propuesta de solución se emplearon las técnicas:

- **Tormenta de ideas:** Es una técnica de reuniones en grupo cuyo objetivo es la generación de ideas en un ambiente libre de críticas o juicios (Pressman, 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, 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.

2.2.1 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, cumplimiento, seguridad u otra índole (Pressman, 2010).

Una vez realizado el levantamiento de información e identificadas las necesidades del cliente, se identificaron un total de 12 requisitos funcionales. En la siguiente tabla se muestran las descripciones de los mismos, y para una mayor interpretación consultar el documento *"Especificación de requisitos de software"* elaborado por el autor de la presente investigación.

Tabla 1: descripción de los requisitos identificados

Nº	Nombre	Descripción
RF1.	Adicionar alerta	Permite agregar una nueva alerta en dependencia de la instancia
RF2.	Copiar alerta	Permite realizar una copia de una alerta creada por el presidente del tribunal hacia la sala
RF3.	Modificar alerta	Permite realizar cambios en las alertas configuradas
RF4.	Eliminar alerta	Permite eliminar alertas configuradas con anterioridad
RF5.	Activar alerta	Permite cambiar la alerta al estado activado

RF6.	Desactivar alerta	Permite cambiar la alerta al estado desactivado
RF7.	Listar alerta	Permite mostrar un listado con las alertas configuradas
RF8.	Adicionar notificación	Permite adicionar una nueva notificación
RF9.	Listar notificación	Permite listar las notificaciones de la sala o del tribunal, en dependencia de la instancia
RF10.	Visualizar notificación	Permite mostrar el contenido de la notificación
RF11.	Eliminar notificación	Permite eliminar notificaciones configuradas anteriormente
RF12.	Notificar mediante correo electrónico	Permite enviar una notificación vía correo electrónico a los usuarios que forman parte del XEJEL.

Fuente: elaboración propia

2.2.2 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, 2010).

El equipo de desarrollo de XEJEL identificó un total de 13 RnF, a continuación, se describen cada uno de estos:

Usabilidad

RnF 1. Requisito de usabilidad 1

En el sistema se deben visualizar todos los mensajes en idioma español. La tipografía debe ser uniforme, de un tamaño adecuado.

RnF 2. Requisito de usabilidad 2

El sistema debe facilitar la interacción con el usuario, posibilitando la utilización de combinaciones de teclas, podrán utilizarse los campos de selección en la interfaz en los casos que sea posible y se agruparán los vínculos y botones por grupos funcionales siempre que se cumpla con las pautas de diseño de las interfaces.

RnF 3. Requisito de usabilidad 3

El sistema debe ofrecer una interfaz amigable, fácil de operar. Igualmente debe mantener la línea de diseño establecida la cual mantiene la uniformidad y representatividad de la solución. Las interfaces deben poseer un diseño sencillo, con pocas entradas, permitiendo un balance adecuado entre funcionalidad y simplicidad de tal manera que no se haga difícil para los usuarios la utilización del

mismo.

Confiabilidad

RnF 4. Requisito de confiabilidad 1

El sistema debe permitir la visualización de la información necesaria para advertir cualquier excepción en el mismo. La información detallada será almacenada en registros de los diferentes componentes de la aplicación.

Eficiencia

RnF 5. 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 3 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 10 segundos.

Restricciones del diseño y la implementación

RnF 6. Requisito de restricción de diseño e implementación 1

La PC[16] cliente del sistema debe poseer resolución de 1024 por 768 pixeles o superior.

RnF 7. Requisito de restricción de diseño e implementación 3

El sistema se debe desarrollar utilizando el lenguaje de programación del lado del servidor: PHP 7.2 o superior. Lenguaje de programación del lado del cliente: HTML5, CSS3, Javascript y Typescript. Debe estar instalado en el servidor *Alternative PHP Cache*, así como todas las bibliotecas y configuraciones de las que depende el funcionamiento correcto de Symfony3 y PostgreSQL 10 o superior.

Interfaz de usuario

RnF 8. Requisito de interfaz de usuario 1

El sistema debe estar disponible durante el horario laboral. En caso de ser necesaria una actualización al software, esta se hará mayormente en horario no laboral, en caso contrario, la misma no debe durar más de 24 horas, teniendo en cuenta que previamente se deben haber validado y probado las modificaciones realizadas.

RnF 9. Requisito de interfaz de usuario 2

Se usarán los siguientes widgets en los formularios: *checkbox* cuando se tiene menos de 4 elementos

[16] Computadora personal (**PC** por sus siglas en inglés de *Personal Computer*)

posible a seleccionar y se pueden seleccionar varios, en caso de que se deba seleccionar solo uno, se usará *radio button*; *select* para seleccionar elementos de una lista en las que existan más de 3 posibles elementos; *textfield* para los campos de texto y *textarea* para textos de mayor longitud. Todos los campos tendrán un *label*, el mismo se encontrará encima del *widget* y contendrán asteriscos (*) de color rojo en caso de ser obligatorio el campo, los mensajes de validaciones irán de color rojo y se mostrarán debajo de *widget* y de igual forma, pero de un color más opaco se mostrará algún mensaje de ayuda del campo.

RnF 10. Requisito de interfaz de usuario 3

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.

RnF 11. Requisito de interfaz de usuario 4

Los botones tendrán dos tonalidades de colores, los de acciones primarias, que serían aquellas acciones que terminan o completan una acción, lo que siempre se quiere lograr, serían los que llamen la atención a la vista y los secundarios, que son las acciones que se pueden hacer, pero no es lo que debe de ocurrir normalmente, ejemplo un cancelar, irán pintados de un color más claro.

RnF 12. Requisito de interfaz de usuario 5

Se debe configurar un sistema de respaldo ante fallos del servidor que garantice una copia de la aplicación, así como del estado diario de la base de datos.

2.2.3 Validación de los requisitos

El resultado del trabajo realizado es una consecuencia de la Ingeniería de Requisitos (especificación del sistema e información relacionada) y es evaluada su calidad en la fase de validación. 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, 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, 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. En el epígrafe 2.2.4 se

hace referencia a las Historias de Usuario como producto de trabajo de la disciplina de Requisitos, en la misma se detallan los prototipos obtenidos para la propuesta de solución.

- **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, 2011). En el Capítulo 3 del presente documento se relacionan los casos de pruebas generados en esta técnica.
- **Revisiones de los requisitos:** se realizaron revisiones de cada requisito, comprobando que los mismos no presenten errores e inconsistencias (Sommerville, 2011). Esta técnica se llevó a cabo por parte de un equipo de personas, validando que la interpretación de cada una de las descripciones no sea ambigua, ni presenten omisiones o errores, evaluando así la calidad de la especificación de cada uno. En el anexo 3 del presente documento se relacionan los acuerdos tomados en estas revisiones.

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:

$$Nr = 12 + 12$$

$$Nr = 24$$

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, de los requisitos especificados para el desarrollo de la propuesta de solución, dos de ellos causaron contradicción (RF2 y RF10) en sus interpretaciones. Por tanto, la variable Q1 obtiene el siguiente valor:

Q1 = 22/ 24

Q1 = 0.92

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 0.92, se concluye que el 92% de los requisitos se obtuvieron una sola interpretación. Los dos requisitos identificados como ambiguos fueron modificados y validados para garantizar su correcta interpretación, llegando al resultado ideal de Q1=1.

2.2.4 Historia de usuario

Las historias de usuario (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 identificaron 12 HU, las cuales se pueden consultar en el documento *"Historias de usuario de los componentes de alertas y notificaciones para el XEJEL"*, creado por el autor de la presente investigación. A continuación, se describe una HU de prioridad Alta para el cliente:

Tabla 2: HU_Adicionar alerta

Número: 1	**Requisito:** Adicionar alerta
Programador: Omar Sierra Sarduy	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:** 8 horas
Riesgo en Desarrollo:	**Tiempo Real:** 8 horas

Descripción: Permite agregar una nueva alerta en dependencia de la instancia.

Campos:

- **Cantidad de días:** Se refiere a la cantidad de días que se deben tener en cuenta al realizar la alerta, que una vez pasado el término teniendo en cuenta esta cantidad de días, el sistema emite una notificación al usuario. Campo numérico. No puede ser nulo.
- **Estado del expediente:** Se refiere a una lista despegable donde se relacionan cada uno de los estados del expediente, que están en dependencia del proceso en el que se encuentra dicho expediente. Campo de selección. No puede ser nulo.

Botones:

Adicionar: Opción que muestra los campos para poder configurar una nueva alerta.

Guardar: Opción que guarda los datos

Cancelar: Opción que descarta los cambios realizados y oculta el área de adicionar las alertas.

Observaciones: N/A

Prototipo elemental de interfaz gráfica de usuario:

Fuente: elaboración propia

2.3 Arquitectura de software

La arquitectura del software de un programa o sistema de cómputo es la estructura o estructuras del sistema, lo que comprende a los componentes del software, 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, 2010).

La solución propuesta tiene como base la arquitectura del XEJEL, separada en *frontend* y *backend.* En la figura 2 se muestran los elementos de esta arquitectura y las partes donde inciden los componentes que engloban la solución propuesta. El *frontend* es la parte del software que interactúa con los usuarios y el *backend* es la parte que procesa la entrada desde el *frontend.* La idea general es que el *frontend* sea el responsable de recolectar los datos de entrada del usuario y los transforma ajustándolos a las especificaciones que demanda el *backend* para poder procesarlos, devolviendo generalmente una respuesta que el *frontend* recibe y expone al usuario de una forma entendible. Para la solución propuesta la conexión entre ambas partes se realiza a través del protocolo HTTP intercambiando datos en formato JSON[17].

[17] (Notación de objetos JavaScript, por sus siglas en inglés JavaScript Object Notation)

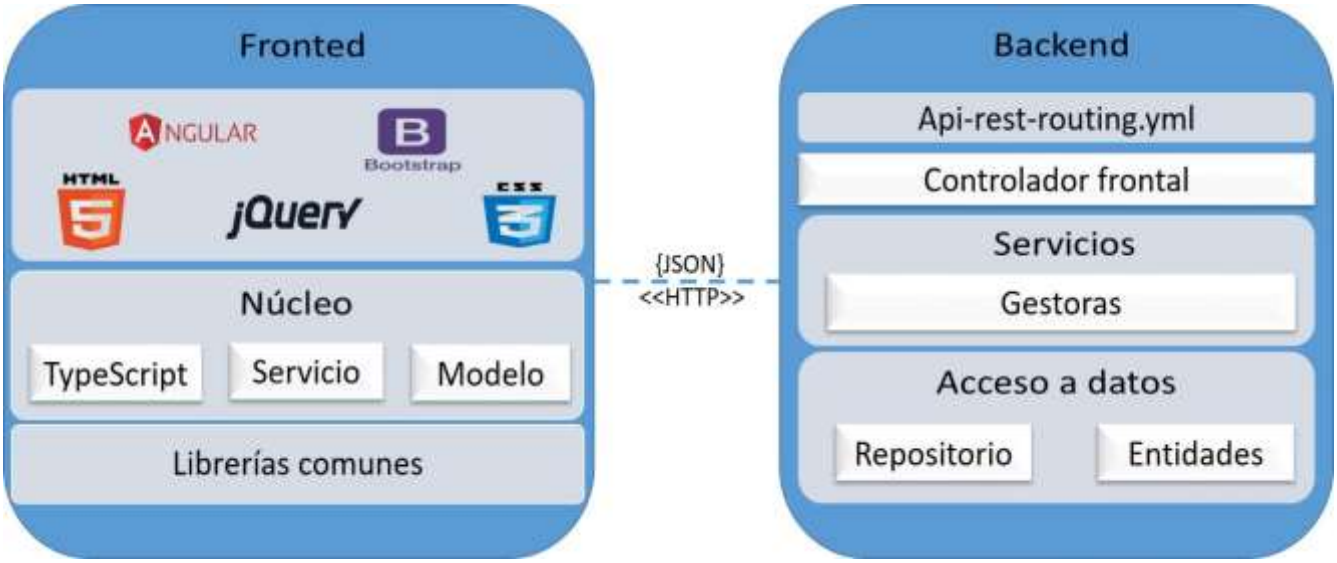


Figura 1: arquitectura de XEJEL

Fuente: elaboración propia

En las figuras 3 y 4 se muestran una descripción detallada de cómo se representa el *frontend* del XEJEL para la arquitectura de la propuesta de solución para el caso de adicionar las alertas y las notificaciones respectivamente, a partir del uso del patrón arquitectónico Modelo Vista Controlador (MVC).

Este patrón separa y desacopla la lógica de negocio de la interfaz del usuario. Para ello divide la aplicación en tres capas: el modelo, la vista y el controlador: (Consultant, 2018).

- El modelo contiene una representación de los datos que maneja el sistema, su lógica de negocio, y sus mecanismos de persistencia.
- La vista, o interfaz de usuario, compone la información que se envía al cliente y los mecanismos de interacción con éste.
- El controlador, actúa como intermediario entre el Modelo y la Vista, gestionando el flujo de información entre ellos y las transformaciones para adaptar los datos a las necesidades de cada uno (ASP.NET, 2017).

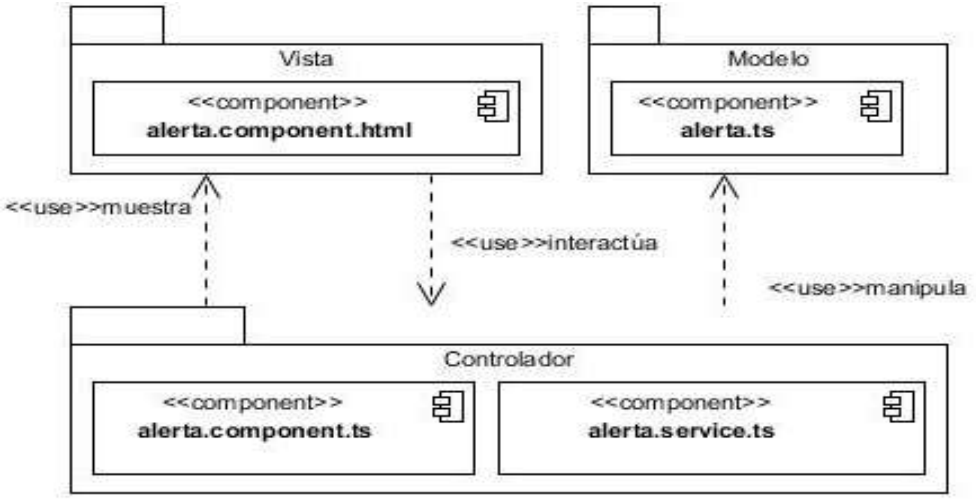


Figura 2: arquitectura *fronted* para el RF_Adicionar alerta

Fuente: elaboración propia

En la figura anteriormente mostrada, se tiene el modelo compuesto por el archivo *alerta.model.ts*, el cual contiene una representación de los datos que maneja el sistema. Por otra parte, la vista se compone por el archivo *alerta.component.html*, este representa la información visible al usuario e interactúa con funciones específicas del controlador. El controlador está compuesto por el archivo *alerta.component.ts*, encargado de interactuar con el modelo y mostrar la información a la vista.

A partir de la figura 2, en la parte del *backend,* la propuesta de solución incide en las capas controladora, gestora y acceso a datos. Estas son implementadas sobre la arquitectura *backend* del XEJEL a través del patrón arquitectónico N Capas.

Una arquitectura de N Capas divide una aplicación en capas lógicas y niveles físicos. Las capas son una forma de separar responsabilidades y administrar dependencias. Cada capa tiene una responsabilidad específica. Una capa superior puede utilizar los servicios de una capa inferior, pero no al revés (Microsoft Azure, 2018).

En la siguiente figura se muestran una descripción detallada de cómo se representa el *backend* del XEJEL para la arquitectura de la propuesta de solución en el caso del RF_Adicionar las alertas.

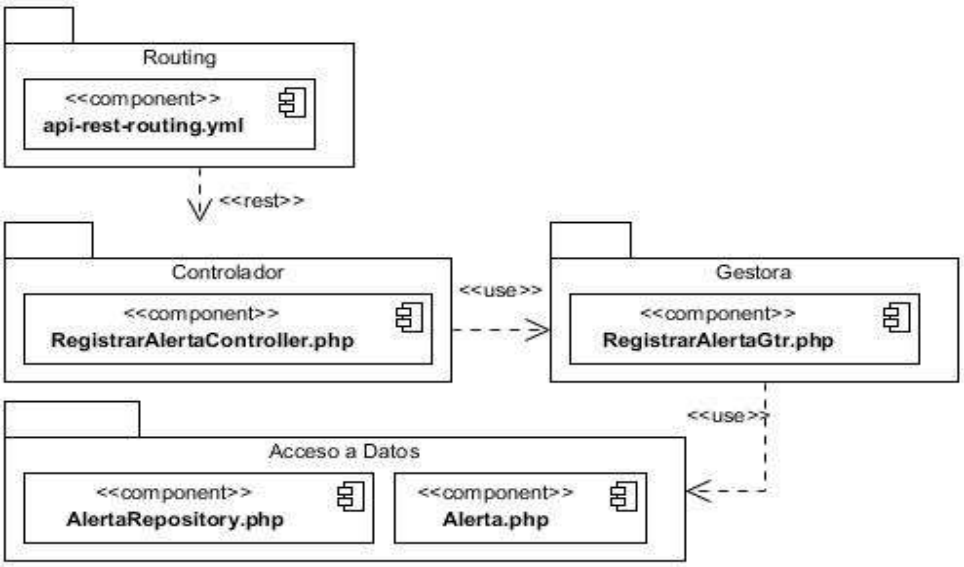


Figura 3: arquitectura *backend* para el RF_Adicionar alerta

Fuente: elaboración propia

En las figuras anteriormente mostradas, la clase *RegistrarAlertaController* de la capa Controladora usa la clase de la capa Gestora *RegistrarAlertaGtr*, comunica con la base de datos, específicamente con la clase *Alerta.php* a través de su repositorio de datos: *AlertaRepository*. La relación entre los componentes del *frontend* y del *backend* se ve reflejada en el componente *api-rest.routing.yml.*

2.4 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, 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:

2.4.1 Patrones de diseño GRASP[18]

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). Este patrón se evidencia en la solución a través de las clases controladoras que se encuentran en la carpeta Controller en el *backend*, y en la

[18] General Responsibility Assignment Software Patterns (Patrones de software de asignación de responsabilidad general)

estructura de Symfony3 se evidencia en el uso del controlador frontal que se encuentra en la carpeta web de cada proyecto creado con dicho marco de trabajo. En la figura que se muestra a continuación se evidencia este patrón al crear la clase *AlertaController*:

```php
/** Alerta ...*/
class Alerta {

    /** @var int ...*/
    private $id;

    /** @var int ...*/
    private $cantidadDias;

    /** @var bool ...*/
    private $activo;

    /** @var int ...*/
    private $idobjetoAlerta;

    /** @ORM\ManyToOne(targetEntity= "AppBundle\Entity\Comun\TipoAlerta") ...*/
    private $tipoAlerta;

    /** @ORM\ManyToOne(targetEntity= "AppBundle\Entity\Comun\TipoEstadoExpediente") ...*/
    private $tipoEstadoExpediente;

    /** @ORM\OneToMany(targetEntity= "AppBundle\Entity\Comun\Notificacion", mappedBy="alerta", cascade={"all"}, orphanRemoval=true) ...*/
    private $notificaciones;

    /** Constructor ...*/
    public function __construct(){...}
```

Figura 4: patrón Controlador. Clase *AlertaController*

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). Este patrón se pone en práctica en la solución en las clases entidades que son las expertas en información y las encargadas de manejar la lógica del negocio que comprenden cada uno de los conceptos que engloban en la propuesta de solución. En la siguiente figura se muestra un fragmento de la clase entidad *Alerta.php*, la cual es la experta en información para el tratamiento de las alertas.

```php
/** Alerta ...*/
class Alerta {

    /** @var int ...*/
    private $id;

    /** @var int ...*/
    private $cantidadDias;

    /** @var bool ...*/
    private $activo;

    /** @var int ...*/
    private $idobjetoAlerta;

    /** @ORM\ManyToOne(targetEntity= "AppBundle\Entity\Comun\TipoAlerta") ...*/
    private $tipoAlerta;

    /** @ORM\ManyToOne(targetEntity= "AppBundle\Entity\Comun\TipoEstadoExpediente") ...*/
    private $tipoEstadoExpediente;

    /** @ORM\OneToMany(targetEntity= "AppBundle\Entity\Comun\Notificacion", mappedBy="alerta", cascade={"all"}, orphanRe
    private $notificaciones;

    /** Constructor ...*/
    public function __construct(){...}

    /** Metodos get y set de los atributos ...*/
}
```

Figura 5: patrón Experto. Clase entidad *Alerta.php*

Fuente: elaboración propia

Patrón Creador: Permite identificar quién debe ser el responsable de la creación (o instanciación) de nuevos objetos o clases (Larman, 2016). Este patrón se evidencia en la propuesta de solución en las clases gestoras (*NombreClaseGtr.php*) en el momento de crear nuevas instancias de las clases entidades. En la figura que se muestra a continuación, este patrón se evidencia, en la clase gestora AlertaGtr.php al crear las instancias de la clase entidad encargada de gestionar toda la información referente a las alertas:

```php
$alerta = new Alerta();
$alerta->setCantidadDias($alert['cantidad_dias']);
$alerta->setTipoAlerta($this->getEm()->find(TipoAlerta::class, $this->obtenerTipoAlerta($tipo)));
$alerta->setTipoEstadoExpediente(
    $this->getEm()->find(TipoEstadoExpediente::class, $alert['tipo_estado_expediente']['id'])
);
$alerta->setIdobjetoAlerta($idobjeto);
$alerta->setActivo($alert['activo']);
```

Figura 6: patrón Creador. Instancia de la entidad *Alerta.php*

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, 2016). Symfony favorece ampliamente el bajo acoplamiento de las clases en el sistema, dado que a cada clase se le asignan solamente las responsabilidades necesarias de manera que no dependan en gran medida de otras (ver Figura 1).

Patrón Alta cohesión: Este patrón permite asignar responsabilidades de manera que la cohesión siga siendo alta (Larman, 2016). El marco de trabajo Symfony favorece la alta cohesión asignando responsabilidades a las clases de tal manera que estas se encuentren estrechamente relacionadas entre sí y no lleguen a realizar un trabajo excesivo (ver Figura 1).

Patrón Fabricación pura: Define una interfaz para crear un objeto, pero deja que sean las subclases quienes decidan qué clase instanciar; su objetivo es devolver una instancia de múltiples tipos de objetos, que normalmente provienen de una misma clase padre y solo se diferencian entre ellos por algún aspecto de comportamiento. Las clases de fabricación pura casi siempre se dividen atendiendo a su funcionalidad, dicho con otras palabras, se confeccionan clases destinadas a conjuntos de funciones (Larman, 2016). Este patrón se evidencia en las clases útiles (*NombreClaseUtil.php*) implementadas en el XEJEL las cuales forman parte de los servicios de la aplicación. A continuación, se muestra la clase *EstructuraUtil.php*, encargada de brindar los servicios en cuanto al tribunal y sala que se encuentra activa en el sistema:

```php
/**
 * Class EstructuraUtil
 * @package AppBundle\Core\Util
 */
class EstructuraUtil
{
    const TRIBUNAL_ACTUAL = 'tribunal-actual';
    const SALA_ACTUAL = 'sala-actual';

    /** Retorna el id del tribunal en el que se esta trabajando     * ...*/
    public static function getTribunalActual(){...}

    /** Retorna el id de la sala en la que se esta trabajando     * ...*/
    public static function getSalaActual(){...}

    /** Retorna si se esta trabajando en una sala, o aun no se he antrado al alguna ...*/
    public static function isInSala(){...}

    /** Metodo que retorna la header de key, la que se especifica en el parametro ...*/
    private static function getValue($key){...}
}
```

Figura 7: patrón Fabricación Pura. Clase *EstructuraUtil.php*

Fuente: elaboración propia

2.4.2 Patrones de diseño GoF[19]

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. La aplicación de este patrón se evidencia en la siguiente figura:

[19] Gang Of Four (Pandilla de cuatro)

```
@Component({
    selector: 'app-crear-alerta',
    templateUrl: './crear-alerta.component.html',
    styleUrls: ['./crear-alerta.component.scss']
})
```

Figura 8: patrón Creador. Clase *crear-alerta.componet.ts*

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. Es uno de los principales patrones de diseño utilizados en interfaces gráficas de usuario (GUI), ya que permite desacoplar al componente gráfico de la acción a realizar (Larman, 2016). Este patrón es empleado en varias partes de la implementación de los componentes en el *frontend*, ejemplo al enviar los datos de la vista al controlador, este último hace la petición de un servicio a través de una inyección y se queda a la espera de la respuesta que dicho servicio debe proveer para así emitir o no un evento como se muestra en la siguiente figura:

```
onSubmit() {
    this.alertaService.create(this.alerta, this.obtenerTipoAlerta()).subscribe( next: response => {
        if (!this.alerta.id && response.length !== 0) {
            this.alertas.push(response);
            this.notificationService.success( message: 'app.module.alerta.mensajes.generar.alerta');
            this.cancelar();
        } else if (response.id) {
            this.alertas.splice(this.alertaUpdate,  deleteCount: 1);
            this.alertas.push(this.alerta);
            this.alertaUpdate = null;
            this.notificationService.success( message: 'app.module.alerta.mensajes.generar.modificar');
            this.cancelar();
        } else {
            this.notificationService.error('app.module.alerta.mensajes.generar.error');
        }
    });
}
```

Figura 9: patrón Observador. Clase crear-alerta.componet.ts

Fuente: elaboración propia

2.4.3 Otros patrones

Patrón Inyección de dependencias: 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. La aplicación de este patrón se evidencia en la siguiente figura al crear los objetos, de una forma práctica, que serán utilizados en toda la clase:

```
constructor(private alertaService: AlertaService,
            private activatedRoute: ActivatedRoute,
            private notificationService: NotificationsService,
            public alertService: AlertService) {
}
```

Figura 10: patrón Inyección de dependencia. Clase crear-alerta.componet.ts

Fuente: elaboración propia

2.4.4 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, 2016).

A continuación, se muestra el diagrama de clases del diseño para la historia de usuario HU_Adicionar alerta:

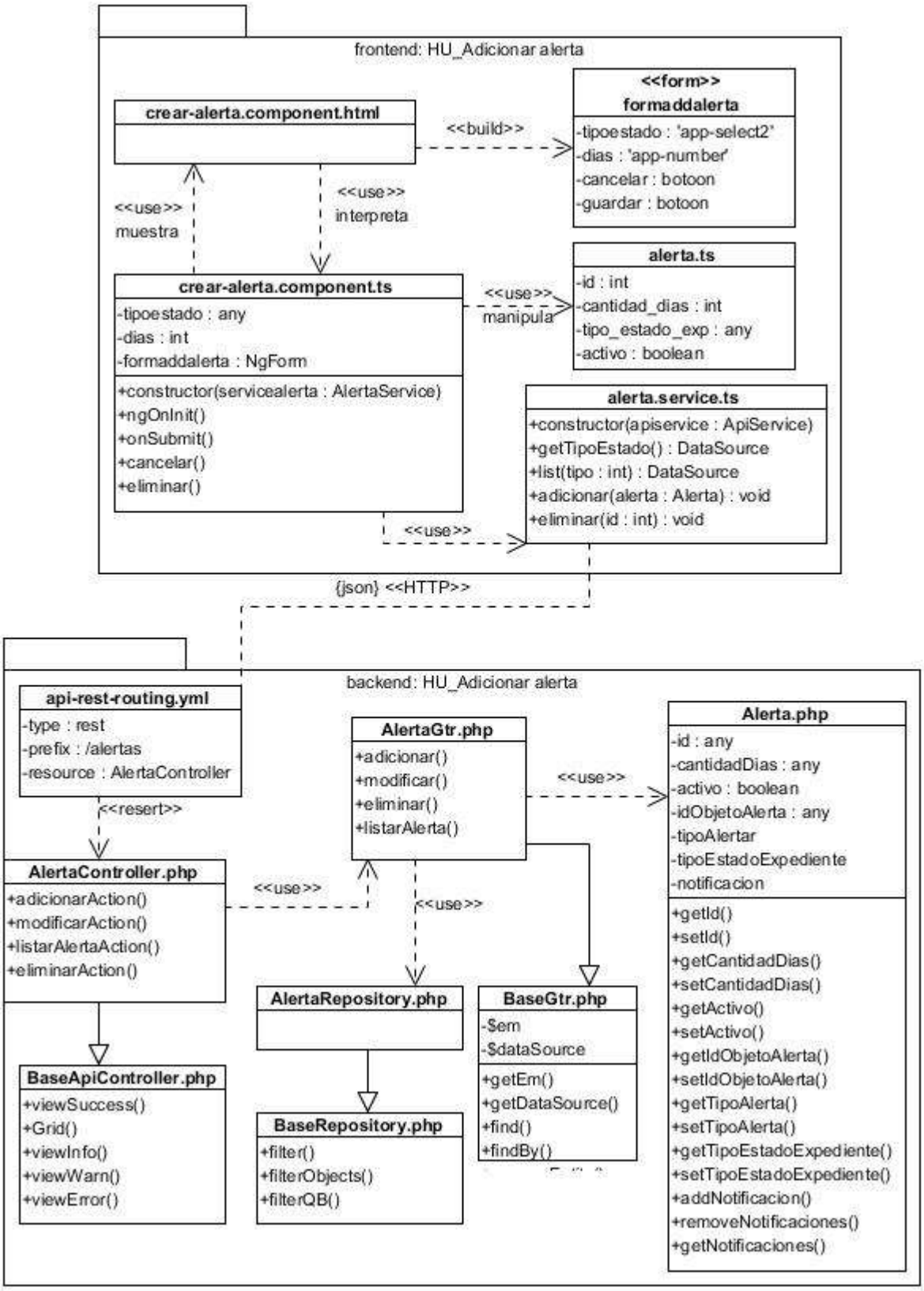


Figura 11: diagrama de clases de diseño para la HU_Adicionar alerta

Fuente: elaboración propia

En el diagrama de clases se muestra la estructura y las relaciones entre clases. Primeramente la vista es la encargada de construir el formulario y de interpretar los datos de la clase *crear-alerta.component.ts*. Esta a su vez es la encargada de mostrar los datos del modelo en la vista obtenidos previamente a través de una petición realizada a la clase *alerta.service.ts* la cual contiene los servicios que se solicitarán al *backend* mediante el protocolo HTTP. La clase *api-rest-routing.yml* permite la comunicación con el *frontend* y redirecciona a la clase controladora, la cual utiliza los servicios de las clases gestoras y esta a su vez manipula las entidades mapeadas previamente desde la base de datos.

2.4.5 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 siguiente figura se muestra el modelo de datos que engloba la estructura lógica y física gestionada al interactuar con los componentes Alerta y Notificaciones en el XEJEL:

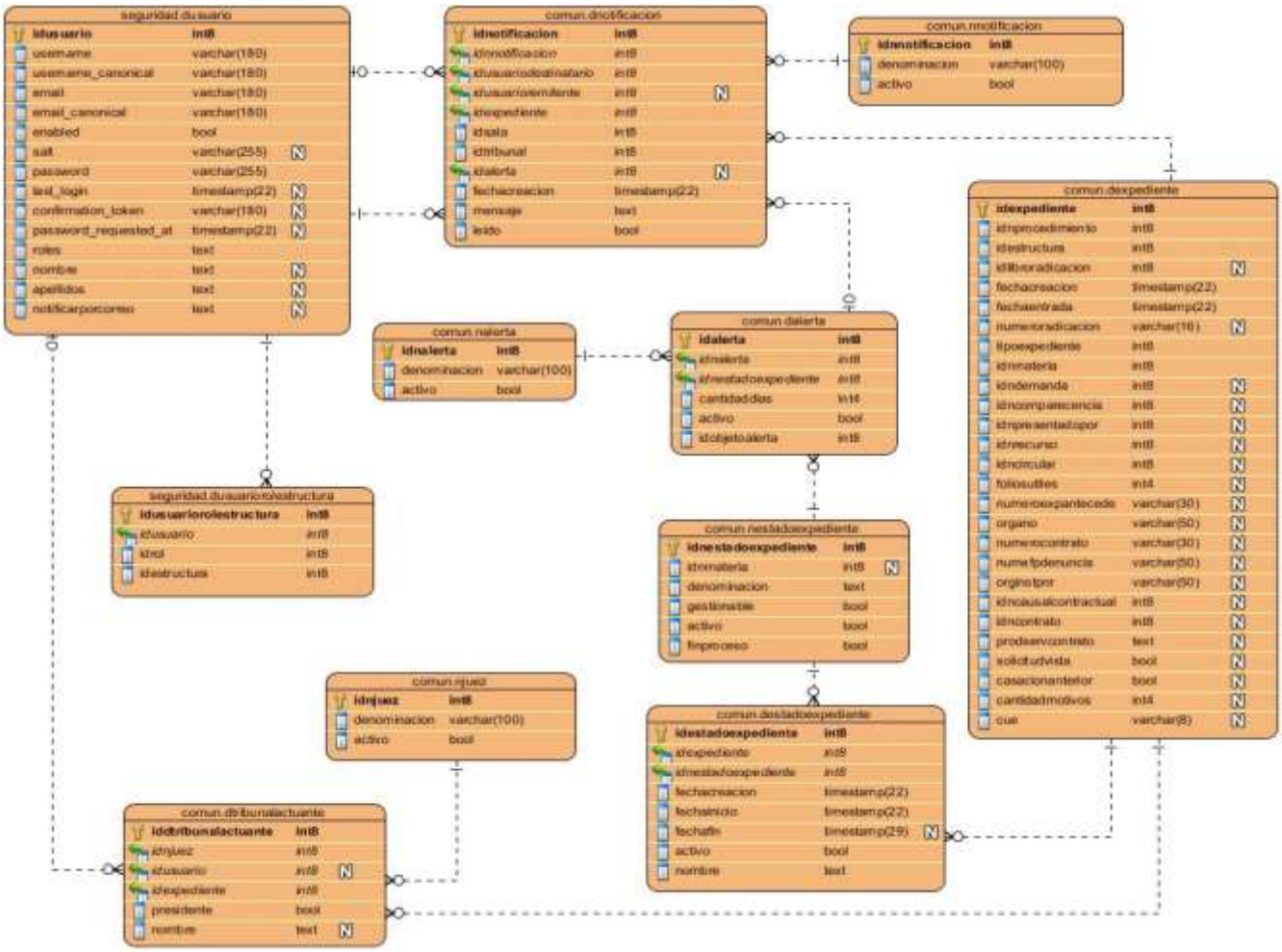


Figura 12: modelo de datos

Fuente: elaboración propia

El modelo de datos obtenido cuenta con un total de 11 tablas, de ellas 4 son nomencladores encargados de gestionar conceptos específicos del negocio anteriormente predefinido, por ejemplo: la tabla *comun.nalerta* se encargará de nomenclar los tipos de alertas que se pueden gestionar en el sistema, dígase: alertas del tribunal y alertas de la sala. Además se cuenta con 7 tablas encargadas de guardar la información referente a cada concepto que intervienen en la solución propuesta, por ejemplo: la tabla *comun.dalerta*, en la cual se guardará la información persistente gestionada en el sistema al interactuar con las alertas del sistema.

2.5 Estándares de codificación

Los estándares de codificación son un estilo de programación homogéneo en un proyecto permite que todos los participantes lo puedan entender en menos tiempo y que el código en consecuencia sea mantenible (Arias Calleja, 2014).

Los mismos fueron definidos por el equipo de arquitectura del proyecto, donde esta información se encuentra plasmada íntegramente en el documento *CEGEL_XEJEL_Estándares_de_codificación_para_PHP.doc*. A continuación, se describen algunos de los estándares más utilizados en la implementación de los componentes para la gestión de Alertas y Notificaciones:

Declaración de clases:

- Las Clases deben ser nombradas de acuerdo a la convención de nombres.
- La llave "{" deberá escribirse siempre en la línea debajo del nombre de la clase ("*one true brace*").
- Cada clase debe contener un bloque de documentación acorde con el estándar de PHPDocumentor.
- Todo el código contenido en una clase debe ser separado con cuatro espacios. Únicamente una clase está permitida por archivo PHP. Incluir código adicional en archivos de clase está permitido pero esta desaconsejado. En archivos de ese tipo, dos líneas en blanco deben separar la clase de cualquier código PHP adicional en el archivo de clase.

A continuación, se muestra un ejemplo de una declaración de clase que es permitida:

```
class AlertaController extends BaseApiController{...}
```

Figura 13: declaración de clases

Fuente: elaboración propia

Funciones y métodos

- Los nombres de funciones pueden contener únicamente caracteres alfanuméricos. Los guiones bajos (_) no están permitidos. Los números están permitidos en los nombres de función, pero no se aconseja en la mayoría de los casos.
- Los nombres de funciones deben empezar siempre con una letra minúscula. Cuando un nombre de función consiste en más de una palabra, la primera letra de cada nueva palabra debe estar en mayúsculas. Esto es llamado comúnmente como formato "camelCase".
- Por norma general, se recomienda la elocuencia. Los nombres de función deben ser lo suficientemente elocuentes como para describir su propósito y comportamiento.

Estos son ejemplos de nombres de funciones admisibles:

```php
public function addAlertaAction($tipo, Request $request, AlertaGtr $alertaGtr)
{
    $alerta = $request->request->all();
    try {...} catch (\Exception $e) {
        if ($e instanceof UniqueConstraintViolationException) {...}
        $error = $e->getMessage();
        $code = $e->getCode();
    }
    return $this->viewError($error, $code);
}
```

Figura 14: funciones y métodos

Fuente: elaboración propia

Constantes: Las constantes pueden contener tanto caracteres alfanuméricos como barras bajas (_). Los números están permitidos. Todas las letras pertenecientes al nombre de una constante deben aparecer en mayúsculas. Las palabras dentro del nombre de una constante deben separarse por barras bajas (_). Por ejemplo:

```php
class StatusCodes
{
    const HTTP_OK = 200; //Cuando funciona correctamente
    const HTTP_CREATED = 201; //Cuando se hace un post para el insert ok
    const HTTP_BAD_REQUEST = 400; //Cuando el Formulario trae erroes
    const HTTP_FORBIDDEN = 403; //NO tiene permiso para relizar la operacion
    const HTTP_NOT_FOUND = 404; //Para cuando no se encuentra lo que se esta buscando
    const HTTP_NOT_ACCEPTABLE = 406; //Cuando lo que se esta haciendo por logica de negocio no es permitido
    const HTTP_CONFLICT = 409;  //Cuando hay algun conflicto de datos repetidos

}
```

Figura 15: definición de constantes

Fuente: elaboración propia

Las constantes deben ser definidas como miembros de clase con el modificador "const". Definir constantes en el alcance global con la función "define" está permitido, pero no recomendado.

Comentarios en las funciones: Todas las funciones deben tener un comentario, antes de su declaración, explicando qué hacen. Ningún programador debería tener que analizar el código de una función para conocer su utilidad. Tanto el nombre como el comentario que acompañe a la función deben bastar para ello.

```php
/**
 *  Metodo para adicionar una alerta en el sistema. Hace referencia al RF.1
 * @Rest\Post("/{tipo}")
 * @param $tipo
 * @Rest\View(serializerGroups={"default", "alerta_tipoestadoexpediente"})
 * @param Request $request
 * @param AlertaGtr $alertaGtr
 * @return \FOS\RestBundle\View\View
 * @throws \Exception
 */
public function addAlertaAction($tipo, Request $request, AlertaGtr $alertaGtr)
```

Figura 16: comentarios en las funciones

Fuente: elaboración propia

Ubicación y denominación de archivos: Se ubicarán los archivos según las convenciones establecidas por Symfony 3.4 o según las especificaciones del equipo de arquitectura. Para la denominación de los archivos se seguirán las convenciones establecidas por Symfony 3.4 o el equipo de arquitectura. Ejemplo:

- Para las clases controladoras se usará el sufijo Controller (ver figura 14)
- Para las clases de gestión del negocio se usará el sufijo Gtr:

```php
class AlertaGtr extends BaseGtr
```

Figura 17: ubicación y denominación. Clase Gtr

Fuente: elaboración propia

- Para las clases que definen las tablas en el sistema se usará el sufijo Tm:

```php
class NotificacionTm extends BaseTableModel
```

Figura 18: ubicación y denominación. Clases TableModel

Fuente: elaboración propia

- Para las clases repositorio se usará el sufijo Repository:

```php
class AlertaRepository extends BaseRepository
```

Figura 19: ubicación y denominación. Clases Repository

Fuente: elaboración propia

Estilo y reglas de escritura de código PHP

- **Nombres de variables:** Los nombres deben ser descriptivos y concisos. No usar ni grandes frases ni pequeñas abreviaciones para las variables. Siempre es mejor saber qué hace una variable con solo conocer su nombre. Esto aplica para los nombres de variables, funciones, argumentos de funciones y clases. Los nombres de las variables y de las funciones pueden inician con letra minúscula, pero si estas tienen más de una palabra, cada nueva palabra debe iniciar con letra

mayúscula. Las constantes deben de escribirse siempre en mayúsculas y tanto estas como las variables globales deben de tener como prefijo el nombre de la clase a la que pertenecen.

- **Definiciones de la función:** Los nombres de la función pueden contener solo caracteres alfanuméricos. Los nombres de la función siempre deben empezar en letras minúsculas. Cuando un nombre de la función consiste de más de una palabra, la primera letra de cada nueva palabra debe capitalizarse (ver figura 15).
- **Llamadas a funciones:** Deben llamarse las funciones sin los espacios entre el nombre de la función, el paréntesis de la apertura, y el primer parámetro; los espacios entre las comas y cada parámetro, y ningún espacial entre el último parámetro, el paréntesis del cierre, y el punto y coma.
- **Siempre incluir las llaves:** En todo momento a la hora de codificar un bloque de instrucciones, este debe ir encerrado entre llaves, aun cuando conste de una sola línea (ver figura 15).
- **No utilizar variables sin inicializar**: Si no se tiene control sobre el valor de una variable, se debe verificar que esté inicializada (ver figura 15).

2.6 Conclusiones del capítulo

La descripción de los componentes para la gestión de alertas y notificaciones, posibilitó lograr un mayor entendimiento del proceso a informatizar y sentar las bases para la disciplina de Requisitos. Asimismo, la especificación de requisitos ayudó a obtener una visión de las funcionalidades a implementar. Por otra parte, la disciplina de Análisis y diseño a través de la descripción de la arquitectura, facilitó establecer una guía para la implementación. Finalmente, los estándares de implementación definidos por parte del equipo de arquitectura del XEJEL, permitieron establecer las pautas de la codificación asegurando la comprensión del código en menos tiempo y que este en consecuencia sea mantenible.

CAPÍTULO 3: VALIDACIÓN DE LA SOLUCIÓN PROPUESTA

En el presente capítulo se define la validación del diseño utilizando las métricas y haciendo un análisis de sus resultados. De igual forma, siguiendo la disciplina de Pruebas internas que propone la metodología AUP-UCI, se verifica el resultado de la implementación desarrollándose pruebas funcionales, unitarias y de rendimiento, esta última para fundamentar cómo la solución propuesta contribuye a la celeridad en el proceso de alertas y notificaciones en los TPC. Luego se procede a la disciplina de Pruebas de liberación para avalar el correcto funcionamiento de los componentes de alertas y notificaciones del XEJEL antes de ser entregados al cliente. Por último, se establecen un conjunto de criterios de medida para la validación de las variables de la investigación.

3.1 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, 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.

3.1.1 Métrica Relación entre clases (RC)

El resultado de la aplicación de la métrica RC está dado por el número de relaciones de uso que se establecen entre una clase y las demás clases existentes. Se evalúa a partir de los siguientes atributos de calidad:

- Acoplamiento: Un aumento de la RC implica un aumento del acoplamiento de la clase.
- Complejidad de mantenimiento: Un aumento de la RC implica un aumento de la complejidad del mantenimiento de la clase.
- Reutilización: Un aumento de la RC implica una disminución en el grado de reutilización de la clase.
- Cantidad de pruebas: Un aumento de la RC implica un aumento de la cantidad de pruebas necesarias para probar una clase (Pressman, 2010).

En las relaciones entre clases, el valor obtenido al aplicar dicha métrica debe ser directamente proporcional al acoplamiento y a la complejidad de mantenimiento; además debe ser inversamente proporcional al nivel de reutilización del código. Para aplicar la métrica RC es necesario categorizar cada una de las clases según la cantidad de relaciones que esta contenga. En la Tabla 3 se muestran las categorías para clasificar cada uno de los atributos de calidad anteriormente mencionados, así como el criterio de evaluación. En la misma se emplean la abreviatura Prom (promedio).

Tabla 3: categoría por atributos y criterio de evaluación. Métrica RC

Atributo	Categoría	Criterio
Acoplamiento	Ninguno	RC = 0
	Bajo	RC = 1
	Medio	RC = 2
	Alto	RC > 2
Complejidad de mantenimiento	Baja	RC <= Prom
	Media	Prom<=RC<= 2* Prom
	Alto	RC >2* Prom
Reutilización	Baja	RC >2* Prom
	Media	Promedio< RC <= 2 *Promedio
	Alto	RC <= Promedio
Cantidad de pruebas	Baja	RC <= Promedio
	Media	Promedio<= RC <2*Promedio
	Alto	RC >= 2*Promedio

Fuente: (Lorenz, y otros, 1994)

La Tabla 4 muestra 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 el 100% de 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 4: resultados obtenidos luego de aplicada la métrica RC

No.	Clase	Cantidad de Relaciones de Uso	Ac	CM	R	CP
1	crear-alerta.component.html	3	Alto	Media	Media	Media
2	formaddalerta	1	Bajo	Baja	Alta	Baja
3	crear-alerta.component.ts	4	Alto	Alta	Baja	Alta
4	alerta.ts	1	Bajo	Baja	Alta	Baja
5	alerta.service.ts	1	Bajo	Baja	Alta	Baja
6	api-rest-routing.yml	1	Bajo	Baja	Alta	Baja
7	AlertaGtr.php	3	Alto	Media	Media	Media
8	Alerta.php	1	Bajo	Baja	Alta	Baja
9	AlertaController.php	2	Medio	Media	Media	Media

10	AlertaRepository.php	1	Bajo	Baja	Alta	Baja
11	BaseGtr.php	0	Ninguno	Baja	Alta	Baja
12	BaseApiController.php	0	Ninguno	Baja	Alta	Baja
13	BaseRepository.php	0	Ninguno	Baja	Alta	Baja
14	adicionarnotificacion.component.html	3	Alto	Media	Media	Media
15	formaddnotificacion	1	Bajo	Baja	Alta	Baja
16	adicionarnotificacion.component.ts	4	Alto	Alta	Baja	Alta
17	notificacion.ts	1	Bajo	Baja	Alta	Baja
18	notificacion.service.ts	1	Bajo	Baja	Alta	Baja
19	Notificacion.Gtr.php	3	Alto	Media	Media	Media
20	Notificacion.php	1	Bajo	Baja	Alta	Baja
21	NotificacionController.php	2	Medio	Media	Media	Media
22	NotificacionRepository	1	Bajo	Baja	Alta	Baja
23	listarnotificacion.component.html	3	Alto	Media	Media	Media
24	table.component.ts	1	Bajo	Baja	Alta	Baja
25	listarnotificacion.component.ts	4	Alto	Alta	Baja	Alta
26	detalles-notificaciones.component.html	3	Alto	Media	Media	Media
27	detalles-notificaciones.component.ts	4	Alto	Alta	Baja	Alta

Fuente: elaboración propia

Después de haber aplicada la métrica, se tomaron todos los resultados obtenidos individualmente y se agruparon para ser analizados de manera general, promediando los valores obtenidos por categoría en cada atributo. En la Figura 20 se muestran estos resultados.

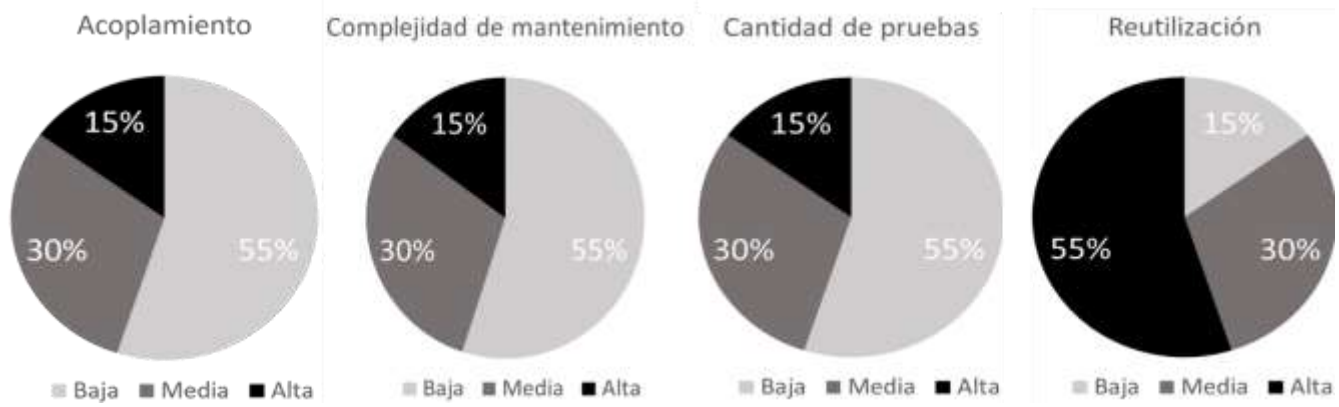


Figura 20: resultado de la métrica RC

Fuente: elaboración propia

Después de evaluar los resultados obtenidos de cada uno de los atributos de calidad que mide esta métrica, se obtuvieron los siguientes resultados:

- El 55% de las clases tienen un porciento medio acoplamiento.
- El 55% baja complejidad de mantenimiento y un porciento bajo de cantidad de pruebas.
- El 55% de las clases poseen una alta reutilización.

Como se puede observar en la figura el acoplamiento posee un nivel bajo por lo que existe poca dependencia entre las clases trayendo como consecuencia una alta probabilidad de reutilización. También se puede apreciar que existe un bajo nivel de complejidad de mantenimiento por lo que a la hora de optimizar métodos y demás operaciones no es necesario realizar una gran cantidad de pruebas.

3.1.2 Métrica Tamaño Operacional de las Clases (TOC)

Las métricas basadas en el Tamaño Operacional de las Clases se basan en contar la cantidad de atributos y la cantidad de operaciones que tiene una clase individual y el promedio que presenta el sistema en su totalidad. Se evalúa a partir de los siguientes atributos de calidad:

- Responsabilidad
- Complejidad de implementación
- Reutilización

Una vez analizado el indicador tamaño de clase, si el valor resultante tiende al crecimiento, es probable que la clase posea un alto grado de Responsabilidad; en consecuencia, el nivel de Reutilización sería mínimo y la implementación altamente compleja. Para medir los atributos de calidad se definieron los umbrales que se muestran en la Tabla 5. En la misma se emplean la abreviatura Prom (promedio).

Tabla 5: categoría por atributos y criterio de evaluación. Métrica TOC

Atributo de calidad	Categoría	Criterio
Responsabilidad	Baja	TOC < =Promedio (Prom).
	Media	TOC Entre Prom. y 2* Prom.
	Alta	TOC >2*Prom.
Complejidad de implementación	Baja	TOC < =Prom.
	Media	Entre Prom. y 2* Prom.
	Alta	TOC >2*Prom.
Reutilización	Baja	TOC >2*Prom.
	Media	TOC Entre Prom. y 2* Prom.
	Alta	TOC < =Prom.

Fuente: (Lorenz, y otros, 1994)

La Tabla 6 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 abreviaturas Rp (Responsabilidad), CI (Complejidad de implementación) y R (Reutilización).

Tabla 6: resultados obtenidos luego de aplicada la métrica TOC

No.	Clase	Cantidad de Procedimientos	Rp	C	R
1	crear-alerta.component.html	8	Baja	Baja	Alta
2	formaddalerta	3	Baja	Baja	Alta
3	crear-alerta.component.ts	14	Media	Media	Media
4	alerta.ts	1	Baja	Baja	Alta
5	alerta.service.ts	15	Media	Media	Media
6	api-rest-routing.yml	2	Baja	Baja	Alta
7	AlertaGtr.php	6	Baja	Baja	Alta
8	Alerta.php	14	Media	Media	Media
9	AlertaController.php	6	Baja	Baja	Alta
10	AlertaRepository.php	1	Baja	Baja	Alta
11	BaseGtr.php	7	Baja	Baja	Alta
12	BaseApiController.php	6	Baja	Baja	Alta
13	BaseRepository.php	25	Alta	Alta	Baja
14	adicionarnotificacion.component.html	4	Baja	Baja	Alta
15	formaddnotificacion	3	Baja	Baja	Alta
16	adicionarnotificacion.component.ts	5	Baja	Baja	Alta
17	notificacion.ts	1	Baja	Baja	Alta
18	notificacion.service.ts	1	Baja	Baja	Alta
19	Notificacion.Gtr.php	13	Media	Media	Media
20	Notificacion.php	21	Alta	Alta	Baja
21	NotificacionController.php	10	Media	Media	Media
22	NotificacionRepository	2	Baja	Baja	Alta
23	listarnotificacion.component.html	1	Baja	Baja	Alta
24	table.component.ts	4	Baja	Baja	Alta
25	listarnotificacion.component.ts	4	Baja	Baja	Alta

26	detalles-notificaciones.component.html	1	Baja	Baja	Alta
27	detalles-notificaciones.component.ts	1	Baja	Baja	Alta

Fuente: elaboración propia

Al aplicar la métrica TOC a estas 27 clases se obtuvieron para cada atributo de calidad los siguientes resultados:

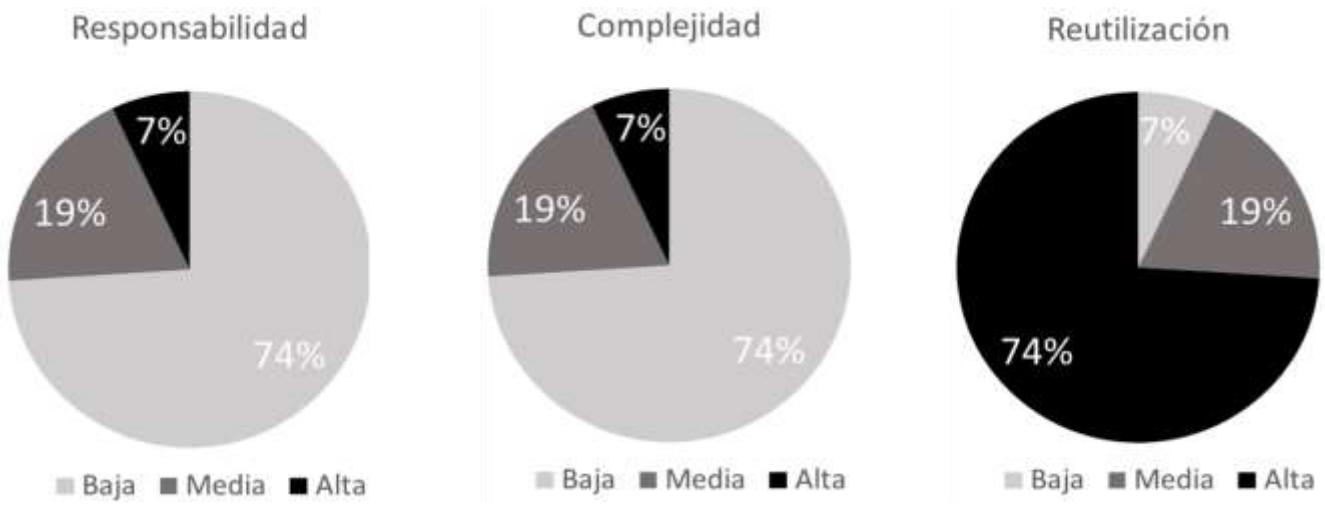


Figura 21: resultado de la métrica TOC

Fuente: elaboración propia

Después de evaluar los resultados obtenidos de cada uno de los atributos de calidad que mide esta métrica, se obtuvieron los siguientes resultados:

- El 74% de las clases poseen una baja responsabilidad y complejidad de pruebas.
- El 74% de las clases poseen una alta reutilización.

Estos resultados demuestran la calidad del diseño de la solución propuesta, ya que, al obtener bajos índices de responsabilidad y complejidad, unidos a un alto grado de reutilización de las clases se facilita en gran medida la implementación de las mismas.

3.2 Pruebas internas

Las pruebas de software representan el porcentaje alto de esfuerzo técnico en el proceso de software. Sin importar el tipo de software que se construya, una estrategia para planificar, ejecutar y controlar pruebas sistemáticas comienza por considerar pequeños elementos del software y moverse hacia afuera, hacia el programa como un todo. El objetivo de las pruebas del software es descubrir errores. Para software convencional, este objetivo se logra mediante una serie de pasos de prueba. Las pruebas de unidad e integración se concentran en la verificación funcional de un componente y en la

incorporación de componentes en una arquitectura de software. Las pruebas de validación demuestran la conformidad con los requerimientos del software y las pruebas del sistema validan el software una vez que se incorporó en un sistema más grande. Cada paso de la prueba se logra a través de una serie de técnicas de prueba sistemáticas que auxilian en el diseño de casos de prueba. Con cada paso de prueba, se amplía el nivel de abstracción con la que se considera el software (Pressman, 2010).

3.2.1 Pruebas unitarias

Las pruebas unitarias se realizan sobre las funcionalidades internas de un módulo y se encargan de comprobar los caminos lógicos, ciclos (bucles) y condiciones que debe cumplir el programa (Pressman, 2010).

Para aplicar las pruebas unitarias, el autor de la presente investigación define utilizar el método de Caja blanca. Con el fin de desarrollar casos de prueba que garanticen la ejecución, al menos una vez, de los caminos independientes por los cuales transitan los procedimientos en la implementación de la solución propuesta. Para aplicar este método se define la técnica de ruta básica que a continuación se describe:

Técnica de ruta básica

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, 2010).

Pressman propone como estrategia para aplicar la ruta básica, realizar un análisis de la complejidad ciclomática de cada procedimiento que componen las clases del sistema; una vez concluido este paso se selecciona el método con valor de contener errores, además de que ofrece una medida del número de pruebas que deben diseñarse para validar la correcta implementación de una determinada función. Esta métrica se calcula sobre un grafo y se puede realizar mediante tres formas distintas:

1. **V(G)** = R.
2. **V(G)** = E – N + 2
3. **V(G)** = P + 1

Conociendo que:

- G: Grafo de flujo (grafo).
- R: El número de regiones contribuye a estimar el valor de la complejidad ciclomática.
- E: Número de aristas.
- V(G): Complejidad ciclomática.
- N: Número de nodos del grafo.

- P: Número de nodos predicados[20] incluidos en el grafo.

Una vez calculada la complejidad ciclomática, el valor obtenido representa el límite superior de pruebas que deberán aplicarse (Pressman, 2010).

En la siguiente figura se muestra el método *addAlertaAction()* de la clase *AlertaController.php* que pertenece al requisito Adicionar alerta. Se selecciona el mismo porque es el método principal de este requisito, ya que es la acción donde se registrar cada una de las alertas que se configuran en el sistema.

```
public function addAlertaAction($tipo, Request $request, AlertaGtr $alertaGtr)
{
    $alerta = $request->request->all();                                    1
    try {
        $alert = $alertaGtr->insert($alerta, $this->obtenerIdObjeto($tipo), $tipo);   2
        return $this->view($alert, StatusCodes::HTTP_CREATED);             3
    } catch (\Exception $e) {                                              4
        if ($e instanceof UniqueConstraintViolationException) {            5
            return $this->view(array(), StatusCodes::HTTP_OK);             6
        } else {                                                           7
            $error = $e->getMessage();
            $code = $e->getCode();                                         8
            return $this->viewError($error, $code);
        }
    }                                                                      9
}                                                                          10
```

Figura 22: método *addAlertaAction()* utilizado para aplicarle la prueba de caja blanca.

Fuente: elaboración propia

A continuación, se realiza el grafo de flujo para el método previamente descrito:

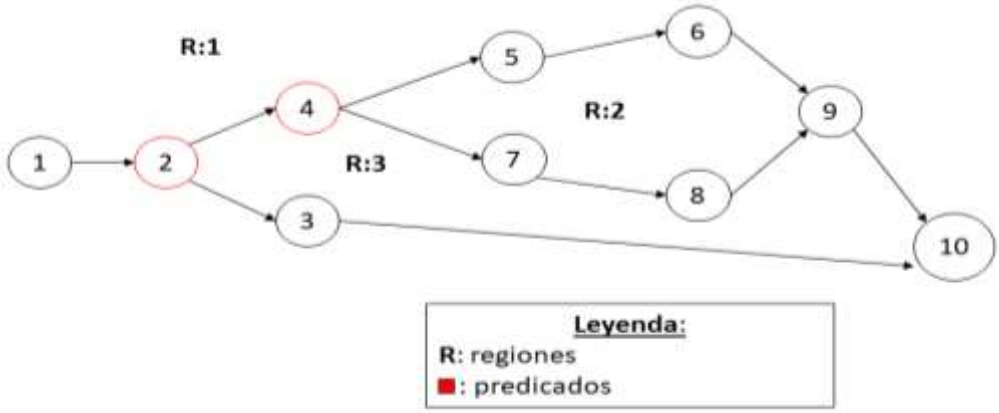


Figura 23: grafo de flujo a partir del método *addAlertaAction()*

Fuente: elaboración propia

[20] Está caracterizado porque dos o más aristas emergen de él

A dicho grafo de flujo se le aplicó la métrica de complejidad ciclomática, calculada por las tres vías conocidas, obteniéndose los siguientes resultados:

- **V(G)** = R = 3.
- **V(G)** = 11 – 10 + 2 = 3.
- **V(G)** =2 + 1 = 3.

Después de calcular la complejidad del grafo, se pudo comprobar que los resultados obtenidos son iguales a 3; por tanto, se deben realizar 3 casos de pruebas, uno por cada ruta independiente. Las rutas independientes (**RI**) resultantes fueron:

RI_1: 1-2-3-10

RI_2: 1-2-4-7-8-9-10

RI_3: 1-3-4-5-6-9-10

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 3 rutas independientes, que dan lugar a la confección de igual número de casos de pruebas, para aplicar las pruebas a este método. A continuación, se muestran el caso de prueba para cada ruta respectivamente:

Tabla 7: caso de prueba de la ruta independiente 1

Caso de prueba: Ruta independiente #1	
Descripción: La alerta que se desea crear es nueva en el sistema por lo tanto se adiciona correctamente.	
Entrada	Cantidad de días y el estado del expediente.
Condición de ejecución	Se desea adicionar una alerta nueva en el sistema.
Resultados esperados	Se adiciona una alerta.

Fuente: elaboración propia

Tabla 8: caso de prueba de la ruta independiente 2

Caso de prueba: Ruta independiente #1	
Descripción: La alerta que se desea crear posee el campo de la cantidad de días con un valor mayor que 10, por lo tanto no se adiciona.	
Entrada	Cantidad de días y el estado del expediente.
Condición de ejecución	La alerta posee el campo “Cantidad de días” con valores inválidos

Resultados esperados	Muestra un error indicando que la alerta posee el campo "Cantidad de días" incorrecto.

Fuente: elaboración propia

Tabla 9: caso de prueba de la ruta independiente 3

Caso de prueba: Ruta independiente #1	
Descripción: La alerta que se desea crear ya se encuentra en el sistema, por lo tanto no se adiciona.	
Entrada	Cantidad de días y el estado del expediente.
Condición de ejecución	La alerta a configurar ya se encuentra en el sistema.
Resultados esperados	Muestra un error indicando que la alerta ya se encuentra creada.

Fuente: elaboración propia

Resultados al aplicar la técnica de ruta básica

Para ejecutar cada caso de prueba se realizaron pruebas manuales utilizando un modelo ejecutable al adicionar las alertas en el sistema. En el caso de prueba # 1 se introdujo una alerta nueva en el sistema, de tal forma que al hacer la comprobación esta se adicionara correctamente. Una vez ejecutado este caso de prueba el resultado fue el esperado. En el caso de prueba # 2, se introdujo una alerta con valores no permitidos por la base de datos, de forma tal que se lanzara una excepción de campos inválidos, siendo esta la respuesta del sistema. En el caso de prueba # 3, se introdujo una alerta, que sus datos coincidiera con una alerta ya creada anteriormente, de tal forma que al hacer la comprobación se lanzara una excepción en el sistema avisando que ya existía una alerta configurada con esos datos, para este caso el resultado fue el esperado.

Una vez ejecutados todos los casos de pruebas obtenidos a través de la aplicación de la técnica ruta básica, se concluye que los mismos fueron probados satisfactoriamente, demostrando que todas las rutas de este código se ejecutaron al menos una vez.

3.2.2 Pruebas funcionales

Las pruebas funcionales son pruebas diseñadas tomando como referencia las especificaciones funcionales de un componente o sistema. Se realizan para comprobar si el software cumple las funciones esperadas (Pressman, 2010). Para llevar a cabo estas pruebas se tuvo en cuenta el método de Caja negra que a continuación se describe.

Método de Caja negra

El método de caja negra permite al ingeniero del software obtener conjuntos de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa. La misma no es una alternativa a las técnicas de método de caja blanca. Más bien se trata de un enfoque complementario que intenta descubrir diferentes tipos de errores que los métodos de caja blanca. Estas pruebas permiten encontrar:

- Funciones incorrectas o ausentes.
- Errores de interfaz.
- Errores en estructuras de datos o en accesos a las bases de datos externas.
- Errores de rendimiento.
- Errores de inicialización y terminación (Pressman, 2010).

Para llevar a cabo el método de Caja negra se utiliza la técnica de Partición equivalente que a continuación se describe.

Técnica de prueba: Partición equivalente

Esta técnica divide el dominio de entrada de un programa en clases de datos a partir de las cuales pueden derivarse casos de prueba. El método se esfuerza por definir un caso de prueba que descubra ciertas clases de errores, reduciendo así el número total de casos de prueba que deben desarrollarse. Una clase de equivalencia representa un conjunto de estados válidos y no válidos para las condiciones de entrada (Pressman, 2010).

Para aplicar esta técnica, se deben primeramente realizar el Diseños de casos prueba (DCP) con el objetivo de obtener un conjunto de pruebas que tengan la mayor probabilidad de descubrir los defectos del software. Un DCP es, en ingeniería del software, un conjunto de condiciones o variables bajo las cuales un analista determinará si una aplicación o una característica de éstos es parcial o completamente satisfactoria (Pressman, 2010).

Como primer paso en el DCP, se muestra a continuación la descripción de las variables para el caso del requisito “Adicionar alerta”.

Tabla 10: descripción de las variables para el RF. Adicionar alerta

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
1	Cantidad de días	Campo de texto	No	Campo de texto que admite solo números en el rango de 1 >= x <= 10.
2	Estado del expediente	Campo de selección	No	Campo de selección único que lista los estados de los expedientes y debe permitir

				seleccionar el que se desee para configurar la alerta.

Fuente: elaboración propia

Luego de obtener la descripción de cada una de estas variables, se procede a realizar cada uno de los escenarios a probar en el sistema. Para el caso del requisito en cuestión, se muestran estos escenarios.

Tabla 11: diseño de caso de prueba para el RF. Adicionar alerta

Escenario	Descripción	Cantidad de días	Estado del expediente	Respuesta del sistema	Flujo central
EC 1.1 Adicionar alerta correctamente.	El usuario introduce los datos correctamente y el sistema permite adicionar la alerta.	V 2	V Presentados	1. El sistema adiciona correctamente la alerta y la muestra en la parte inferior de la región central.	Autenticarse en el sistema/ Opción "Alerta" del menú izquierdo/ Adicionar
EC 1.2 Adicionar alerta con campos en blanco.	El usuario deja campos requeridos vacíos y el sistema no adiciona la alerta.	I V 5	V Turnado I	1. El sistema señala el campo en blanco, indicando que hay campos obligatorios vacíos con el mensaje "Este campo es obligatorio". 2. No habilita el botón Guardar.	
EC 1.3 Adicionar alerta con datos no válidos.	El usuario introduce datos incorrectos y el sistema no adiciona la alerta.	I 9999 V 5	V Returnado N/A	1. El sistema señala el campo con error y muestra el mensaje "Por favor, escriba un valor igual o menor que 999." 2. No habilita el botón Guardar. 1. El sistema adiciona correctamente la alerta y la muestra en la parte inferior de la región central.	

Fuente: elaboración propia

Para aplicar los DCP a la solución, se efectuaron un total de 3 iteraciones para poder alcanzar resultados satisfactorios atendiendo al correcto comportamiento del sistema ante diferentes situaciones. En la figura que se muestra a continuación, se realiza un resumen mediante una gráfica con el número total de No conformidades (NC) por iteración.

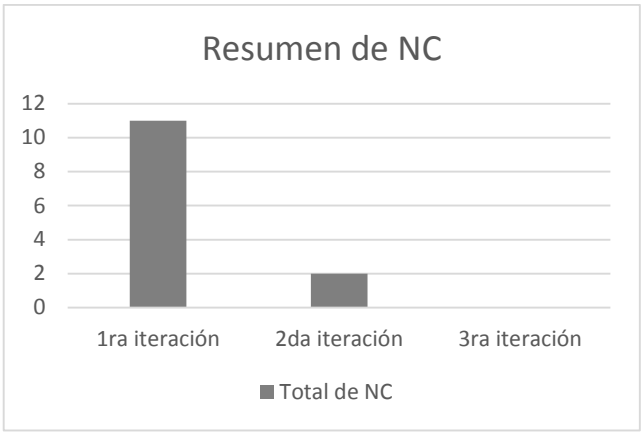


Figura 24: total de NC por iteraciones

Fuente: elaboración propia

Estas NC se clasificaron en: Opciones que no funcionan, Ortografía y Validación. Para la 1ra iteración la tendencia de NC fue de Opciones que no funcionan. En cada una se corrigieron las NC dando paso a que en la tercera iteración no se detectaran NC, lo que trajo consigo que se procediera a la disciplina de pruebas de liberación propuesta por la metodología AUP-UCI. A continuación, se muestra un resumen de las NC por clasificación en cada iteración de prueba:

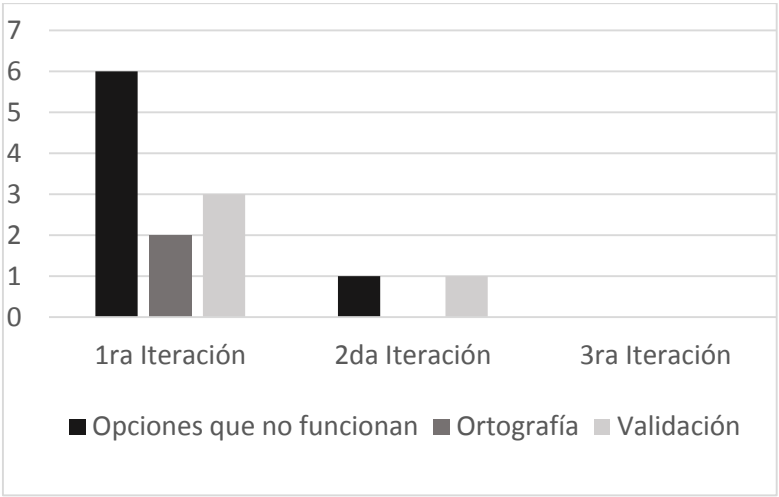


Figura 25: total de NC por clasificación

Fuente: elaboración propia

3.2.3 Pruebas de rendimiento

Las pruebas de rendimiento se diseñan para poner a prueba el rendimiento del software en tiempo de corrida, dentro del contexto de un sistema integrado. Estas ocurren a lo largo de todos los pasos del proceso de prueba. Estas requieren instrumentación de hardware y de software, es decir, con frecuencia es necesario medir la utilización de los recursos, por ejemplo: ciclos del procesador. La

instrumentación externa puede monitorear intervalos de ejecución y eventos de registro conforme ocurren, y los muestreos del estado de la máquina de manera regular (Pressman, 2010).
Para evaluar el tiempo de respuesta del sistema durante las condiciones de carga esperada, y así corroborar la agilización en el proceso “Adicionar alerta”, el autor de la presente investigación define el empleo de la herramienta automatizada JMeter que a continuación se describe.

JMeter

JMeter es un software de código abierto diseñado en Java que permite realizar pruebas de rendimiento y de funcionalidad sobre aplicaciones tipo cliente/servidor escritas en cualquier lenguaje. Se puede utilizar para simular una carga pesada en un servidor, la red o un objeto para poner a prueba su resistencia o para analizar el rendimiento global en diferentes tipos de carga. Igualmente puede ser utilizado para hacer un análisis gráfico de rendimiento o para probar el comportamiento de diferentes elementos con un gran volumen de carga y concurrencia (Apache-JMeter.tm, 2018).

A continuación, se muestra el resultado al ejecutar las pruebas de Rendimiento al enviar una notificación en el sistema, para evaluar el tiempo de respuesta utilizando esta herramienta.

Etiqueta	# Muestras	Media	Mediana	Linea de 90%	Mín	Máx	...	Rendimiento	Kb/sec
/api/notificacion/litigantes-...	5	13	14	14	11	18	...	3,0/sec	2,2
/api/notificacion/	5	13	13	15	11	16	...	3,0/sec	2,2
/api/notificacion/sendmail	5	14	11	12	11	28	...	3,0/sec	2,2
Total	235	60	14	61	2	1440	...	60,1/sec	6323,4

Figura 26: resultados de la prueba de Rendimiento

Fuente: elaboración propia

3.3 Pruebas de liberación

Esta etapa de prueba es ejecutada 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 Sánchez, 2015).
Para este proceso se utilizaron los mismo DCP ejecutados en las pruebas funcionales durante la disciplina de pruebas internas. Las misma fueron llevadas a cabo por el grupo de Calidad CEGEL, entidad encargada y avalada para de evaluar cada uno de los entregables al cliente. A continuación, se muestra una figura con las iteraciones y la cantidad de No conformidades detectadas:

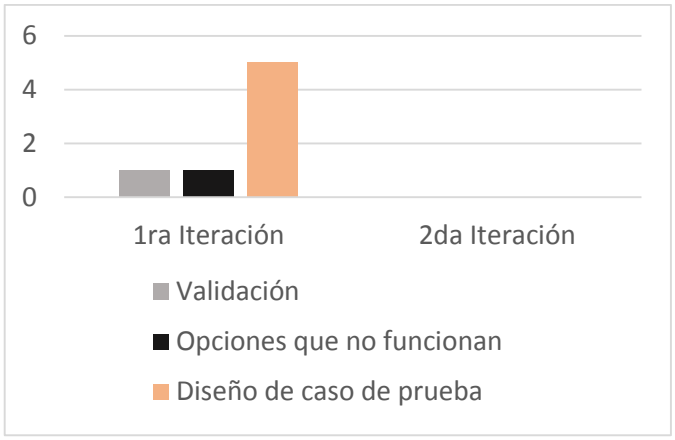


Figura 27: total de NC detectadas en las pruebas de liberación

Fuente: elaboración propia

La liberación del software demostró que los componentes cumplían con las necesidades requeridas y estaban listos para ser entregados al cliente. Luego de terminadas las pruebas se emitió el acta de liberación, corroborando que los componentes estaban listos para ser utilizado por el cliente. En el anexo 5 se puede consultar el acta emitida por esta entidad.

3.4 Validación de las variables de la investigación

Teniendo en cuenta la idea a defender que se plantea en la presente investigación, para analizar la relación causa efecto entre la variable independiente “gestionar las alertas y notificaciones en el Expediente Judicial Electrónico” y la variable dependiente “que contribuya a la celeridad de sus procesos”, el autor de la presente investigación define un conjunto de criterios de medida que permiten validar cómo a través de la propuesta de solución se logra la relación entre ambas variables. La obtención de estos criterios se realiza a partir de las principales deficiencias identificadas en la situación problemática y de los resultados de la encuesta realizada a los especialistas funcionales judiciales de los TPC. En el anexo 4 del presente documento se muestra dicha entrevista.

Criterios de medida definidos:

- ✓ La notificación a las partes de las tramitaciones que se realizan sobre un expediente.
- ✓ Las notificaciones acerca de los vencimientos de los términos de un trámite determinado.
- ✓ Posibilidad de conformar las alertas en los TPC en todas sus instancias y que se le notifique a la parte involucrada.

La evaluación de cada uno de los criterios de medida definidos se muestra a continuación, la misma se realiza a través de un antes (los TPC sin un sistema capaz de gestionar las notificaciones y las alertas) y un después (los TPC con un sistema capaz de gestionar las notificaciones y las alertas), con el propósito de verificar cómo mediante la solución propuesta se contribuye a la celeridad de los procesos

de alerta y notificaciones. Los datos tenidos en cuenta en la comparación fueron tomados a partir de la realización de las pruebas de Rendimiento, comparando el tiempo de respuestas del sistema con el tiempo promedio que se demoran los especialistas para cada una de las actividades que responden a los criterios de medida.

Tabla 12: validación de las variables de investigación

Atributo	Antes	Después
La notificación a las partes de las tramitaciones que se realizan sobre un expediente.	A raíz de la encuesta el 60% de estos especialistas plantean que se tardan entre 1 a 3 días para realizar esta actividad.	Esta actividad utilizando el XEJEL se comprobó que se disminuyó el tiempo a 9 segundos aproximadamente.
Posibilidad de conformar las alertas en los TPC en todas sus instancias y que se le notifique a la parte involucrada	Actualmente los TPC no cuentan con un mecanismo capaz de configurar las alertas que conlleven a una notificación procesal.	Con la utilización del XEJEL se contará con un módulo capaz de configurar las alertas, donde se comprobó que una vez cumplido el tiempo de esa alerta se genera automáticamente una notificación a la parte involucrada.
Las notificaciones acerca de los vencimientos de los términos de un trámite determinado.	A raíz de la encuesta el 50% de estos especialistas plantean que se tardan entre 1 a 3 días para realizar esta actividad.	Esta actividad utilizando el XEJEL se comprobó que automáticamente que se vence el término, se le notifica a las partes involucradas en el proceso sobre dicho vencimiento del término.

Fuente: elaboración propia

La comparación realizada en la tabla anterior, a través de los criterios de medida antes definidos, demuestra cómo, a través de los componentes obtenidos como resultado de la presente investigación, se contribuye a la celeridad de los procesos de alertas y notificaciones en los TPC.

3.5 Conclusiones del capítulo

Las métricas aplicadas para la validación del diseño permitieron cuantificar la calidad del mismo al desarrollar los componentes para la gestión de alertas y notificaciones en XEJEL. Por otra parte, la ejecución de pruebas unitarias para la validación del código, reflejaron que las funcionalidades internas se ejecutan de forma adecuada. De igual forma, la ejecución de pruebas funcionales, evidenció que las funciones externas son operativas, que las entradas se aceptan de forma adecuada y que se producen salidas correctas, obteniendo finalmente una aplicación que satisface los requisitos definidos. Las pruebas de liberación evidenciaron que los componentes cumplen con las necesidades requeridas para ser entregado al cliente. Por último, la relación causa efecto entre la variable independiente y la variable dependiente, a través de varios criterios de medidas definidos, demostró que con la propuesta de solución se contribuye a la celeridad en la gestión de alertas y notificaciones en el XEJEL.

CONCLUSIONES GENERALES

Al concluir la investigación para el desarrollo de los componentes para la gestión de alertas y notificaciones en la herramienta informática Expediente Judicial Electrónico, se tiene como resultado que:

- La elaboración del marco teórico de la investigación mediante el estudio y el análisis de los principales referentes teóricos en los que se sustenta la investigación, facilitó la adquisición de los conocimientos necesarios para el desarrollo de expedientes judiciales electrónicos.
- La identificación de los requisitos, así como el análisis y diseño de la solución propuesta permitió sentar las bases para la implementación de los componentes para la gestión de alertas y notificaciones del XEJEL.
- La implementación de la propuesta de solución contribuyó a la celeridad en el proceso de gestión de alertas y notificaciones del XEJEL.
- La validación de la solución propuesta, aplicando métricas y pruebas de software, permitió avalar con buenos resultados la calidad del diseño y la implementación de los componentes para la gestión de alertas y notificaciones en el XEJEL.
- La validación de las variables de la investigación permitió corroborar cómo se contribuye a la celeridad en la gestión de alertas y notificaciones en el XEJEL

RECOMENDACIONES

Una vez cumplidos los objetivos de la investigación y teniendo en cuenta las experiencias obtenidas en la misma, se recomienda:

- Realizar la integración de los componentes de Alertas y Notificaciones con el resto de los componentes de XEJEL.

REFERENCIAS BIBLIOGRÁFICAS

- **Apache.org. 2018.** The Apache Software Foundation. *The Apache Software Foundation.* [En línea] 2018. http://jmeter.apache.org/.
- **Arias Calleja, Manuel. 2014.** Carmen. Estándares de codificación. 2014.
- **ASP.NET, Servicio de Informatica de. 2017.** MVC 3 Framework. *MVC 3 Framework.* [En línea] 2017. https://si.ua.es/es/documentacion/asp-net-mvc-3/1-dia/modelo-vista-controlador-mvc.html.
- **Castro Morell, Daniel E. y González Guadarrama, José R. 2008.** *Sistema para la tramitación de procesos penales.* 2008.
- **Consultant, Sanchez . 2018.** Sanchez Consultant. *Sanchez Consultant.* [En línea] 2018. https://sanchez-consultant.com/modelo-vista-controlador/.
- **Doctrine.org. 2018.** The Doctrine Project. *The Doctrine Project.* [En línea] 2018. https://www.doctrine-project.org/index.html.
- **Enríquez Ruiz, José Luis, Farías Palacín, Elías y Flores Flores, Eder. 2017.** *Metodología de desarrollo de software.* Rectorado, Universidad Católica Los Ángeles - Chimbote. Chimbote - Perú : s.n., 2017. pág. 39.
- **Enriquez, Yanes , Osmel y Garcia del Busto , Hansel. 2014.** *Mapeo Objeto Relacional.* 2014.
- **Gardey, Julián Pérez Porto y Ana. 2019.** Definición.de. *Definición.de.* [En línea] 2019. https://definicion.de/lenguaje-de-programacion/.
- **—. 2019.** Definición.de. *Definición.de.* [En línea] 2019. https://definicion.de/lenguaje-de-programacion/.
- **Getbootstrap.com. 2018.** Getbootstrap.com. *Getbootstrap.com.* [En línea] 2018. https://getbootstrap.com/.
- **González Ochoa, Darián y Domínguez González, Yosviel. 2018.** *SITPC, una herramienta informática cubana para la administración de la justicia.* La Habana : s.n., 2018.
- **Grados Caballero, Julio Giampiere . 2018.** [En línea] 2018. https://devcode.la/blog/que-es-javascript/.
- **Guevara, Joan. 2017.** [En línea] 2017. https://blog.aulaformativa.com/definicion-usos-ventajas-lenguaje-css3/.
- **Iruela, Juan. 2016.** Revista Digital INESEM. *Revista Digital INESEM.* [En línea] 2016. https://revistadigital.inesem.es/informatica-y-tics/los-gestores-de-bases-de-datos-mas-usados/.
- **IT-CGAE. 2013.** *Manuel de usuario. LEXNET.* s.l. : Red Abogacía Española, 2013. pág. 25.
- **Jiménez Castela, Pedro . 2017.** *Angular 4 desde Cero.* 2017.
- **Juiz, O. 2009.** *INFORME DEL SISTEMA SISECO.* 2009.

- **Juridica, Enciclopedia. 2014.** Enciclopedia Juridica. *Enciclopedia Juridica.* [En línea] 2014. http://www.enciclopedia-juridica.biz14.com/d/diligencias/diligencias.htm.
- **Larman, Craig. 2016.** *UML y Patrones. Una introducción al análisis y diseño orientado a objetos y al proceso unificado.* 3ra . s.l. : Prentice-Hall, 2016.
- **Lorenz, M. y Kidd, J. 1994.** *Object-Oriented Software Metrics.* s.l. : Prentice-Hall, 1994.
- **Lucid_Software_Inc. 2019.** Lucidchart. [En línea] 2019. https://www.lucidchart.com/pages/es.
- **Martin Olivera, Yonnys Pablo y Zamora Sanchez, Gey. 2016.** *ENTORNO DE DESARROLLO INTEGRADO LIBRE Y MULTIPLATAFORMA PARA DESARROLLAR SOFTWARE EDUCATIVO EN FORMATO MULTIMEDIA.* 2016.
- **Menéndez-Barzanallana Asensi, Rafael. 2016.** *Ingeniería del software. .* 2016.
- **Merino, Julián Pérez Porto y María. 2012.** Definicion.de. *Definicion.de.* [En línea] 2012. https://definicion.de/lenguaje-de-programacion/.
- **Microsoft Azure. 2018.** [En línea] 2018. https://docs.microsoft.com/es-es/azure/architecture/guide/architecture-styles/n-tier.
- **MJU, Ministerio de Justicia de España. 2016.** *Expediente judicial electrónico.* 2016.
- **Murua, Juan, Fernández, Alejandro y Eguiluz, Javier. 2019.** *Pro Git, el libro oficial de Git.* 2019.
- **NetBeans.org. 2018.** NetBeans IDE. *NetBeans IDE.* [En línea] 2018. https://netbeans.org/about/index.html.
- **Olarte Gervacio, Luis. 2018.** Conogasi. [En línea] 23 de Abril de 2018. http://conogasi.org/articulos/lenguaje-de-programacion/.
- **Paradigm, Visual. 2017.** *Visual Paradigm.* s.l. : International Ltd, 2017.
- **Pérez Porto, Julián. 2018.** Definición de jQuery. *Definición de jQuery.* [En línea] 2018. https://definicion.de/jquery/.
- **pgAdmin.org. 2016.** The pgAdmin Development Team. *PgAdmin.* [En línea] 2016. https://www.pgadmin.org.
- **Potencier, Fabien y Zaninotto, François. 2008.** *Symfony la guía definitiva.* 2008.
- **Pressman, Roger S. . 2010.** *Ingeniería del software.* 2010.
- **Rodríguez Sánchez, Tamara. 2015.** *Metodología de desarrollo para la Actividad productiva de la UCI.* La Habana : Universidad de las Ciencias Informáticas, 2015.
- **—. 2015.** *Metodología de desarrollo para la Actividad productiva UCI.* La Habana : s.n., 2015.
- **Rouse, Margaret. 2019.** TechTarget, S.A. [En línea] 2019. https://searchdatacenter.techtarget.com/es/definicion/Servidor-Web.
- **Sæther Bakken, Stig, Aulbach, Alexander y Schmid, Egon. 2002.** *Manual de PHP.* 2002.

- **Sanchez Consultant. 2018.** Sanchez Consultant. [En línea] 2018. https://sanchez-consultant.com/modelo-vista-controlador/.
- **Sierra Sarduy, Omar. 2018.** *Centro de Gobierno Electrónico. Especificación de requisitos: XEJEL.* La Habana : UCI, 2018.
- **Sistema-Master-Magazine. 2018.** Sistema-Master-Magazine. [En línea] 2018. https://sistemas.com/herramienta.php#ixzz2qwSpRx1E.
- **Solving Ad Hoc. 2017.** ¿Qué son las historias de usuario y su función en agilidad? [En línea] Diciembre de 2017. https://solvingadhoc.com/las-historias-usuario-funcion-agilidad/.
- **Sommerville, Ian. 2011.** *Ingienería de Software. Novena Edición.* México : PEARSON, 2011.
- **SRL, E. S. J. 2011.** Lex-Doctor, GESTIÓN JURÍDICA. [En línea] 2011. http://www.lex-doctor.com/index.php.
- **W3C. 2018.** W3C. *W3C.* [En línea] 2018. https://www.w3.org/html/.