Facultad CITEC

Tema: Componente para la administración de licenciamiento y colaboración de AUDAT.

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

Autor(es): María del Carmen Hernández Landestoy.

Tutor(es): Ing. Liuba María Infante Infante.

Ing. Miguel Ángel Nápoles Molina.

La Habana, octubre de 2023

Año 64 de la Revolución

DECLARACIÓN DE AUTORÍA

Los autores del trabajo de diploma con título ***"Componente para la administración de licenciamiento y colaboración de AUDAT (Sistema de Auditoría de Datos)"***, concede a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la investigación, con carácter exclusivo. De forma similar se declara a María del Carmen Hernández Landestoy como único autor de su contenido. Para que así conste firman la presente a los 14 días del mes de noviembre del año 2023.

María del Carmen Hernández Landestoy

_____

Firma del Autor

Liuba María Infante Infante

_____

Firma del Tutor

Miguel Ángel Nápoles Molina

_____

Firma del Co-Tutor

DATOS DE CONTACTO

Currículo e información de contacto del **tutor**: Liuba María Infante Infante, Ingeniera en Ciencias Informáticas, lugar de trabajo: Universidad de las Ciencias Informáticas, La Habana, responsabilidades y laborales asumidas: profesora de la asignatura Ingeniería de Software en cursos anteriores y Jefa del proyecto AUDAT, experiencia profesional: 2011-2022, correo electrónico: lminfante@uci.cu.

Currículo e información de contacto del **asesor**: Miguel Ángel Nápoles Molina, Ingeniero en Ciencias Informáticas, lugar de trabajo: Universidad de las Ciencias Informáticas, La Habana, responsabilidades y laborales asumidas: Centro de representación y Análisis de Datos, experiencia profesional: 2022-2023, correo electrónico: mamolina@uci.cu.

AGRADECIMIENTOS

Mis infinitos agradecimientos son para mis padres Maylin y Osmani, ustedes confiaron siempre en mí, como mismo han confiado en cada paso que doy en mi vida, a ustedes me debo.

A mi hermana Naomy por esforzarse tanto en todos estos años, por demostrarme que cuento con ella, sin importar lo que me proponga, ella siempre está ahí.

A mis abuelos, los maternos y paternos (Horacio, Chochi, Rafael y Romelia), los que están y los que ya no, por cuidarme, por estar presentes en mi día a día indistintamente, gracias a ustedes también.

A mis tíos (Mayelin, Rubén, Felicita, Eloy, Ania, David, Milagros, Arnaldo, Kelly, Alied, Tomin, Idania, Luis Alberto, Nancy), a todos gracias infinitas por apoyarme.

A los demás familiares que también me apoyan.

A mis grandes amigos José Carlos y Marielena y por supuesto a sus familias, que son mías también. Ustedes han confiado más que nadie e incluso más que yo y siempre estuvieron seguros de que lo lograría.

A mis tutores de tesis Liuba y Miguel, al tribunal y a la oponente por aconsejarme y ayudarme a hacer de este proceso una de las etapas más duras como estudiante pero también de las más satisfactorias. Gracias.

A mis amigos profesores que también colaboraron en los resultados de este proyecto, Leo, Yurier, Osiel y Liuba.

A los amigos que me ha dado la UCI, con los mismo que he estudiado, compartido, reído, llorado y me he ido de fiesta, y con temor a olvidarme de alguien, los menciono porque tienen un lugar importante en mi corazón: Israel, Isbel, Elizabeth, Amanda, Rey, Eric, Daniela, Fernando, Sachel, Adiel, Lia, Brayan, Ariel, Gealber, Frey, Ernesto, Danilo, La Pitu, Laura, Mardelis, Dayelin, Rody, Raulito, Adrián, Yiyi, Anahomi, Rosmery, Emily, Jennifer, Shaveli,,

Agradecimientos

Melisa, Claudia, Pablo, Power, Aroldo, Alexander, Bladimir, Lauren, Daniel, Dailen, Ciro, Dairon, Fernandito, Enier, Kilmer, Marberci. Y por último pero no menos importante Yadisnet, Yanet, Francy, Pedri y Leo.

A los amigos de mi tierra que desde la distancia también me apoyaron: Lesyani, Rolando, Oile, Yojara, Robislandy, Amanda, Mariangel, Grether, Edilene, Enmanuel, Daniela, Carlos, Omarito, Rosabel, Verónica, Irene, Eddy, Yalena.

DEDICATORIA

A mis padres por guiarme por el buen camino, por impulsarme siempre a ser mejor persona y demostrarme su orgullo en todos los propósitos de mi vida.

A mi hermana por ser el vivo ejemplo del sacrificio, por ser mi apoyo y ayuda incondicional.

A mis abuelos, los que están y los que no están entre nosotros, por cuidarme y demostrarme que SÍ PUEDO.

A mis tíos, los que siempre me han ayudado.

A mis grandes amigos por estar ahí cada día incondicionalmente.

RESUMEN

La Universidad de las Ciencias Informáticas (UCI) mantiene relación a partir de un contrato macro con la Contraloría General de la República de Cuba, donde especialistas del centro CREAD, que radica en la UCI, trabajan en las diferentes versiones del Sistema de Auditoría de Datos (AUDAT). La explotación por los auditores reveló varias inconformidades entre la que destaca la obsoleta tecnología que se está utilizando, además de no poder compartir documentos para una mejor interacción y desenvolvimiento en la tarea. Por ello el presente trabajo de diplomase traza como objetivo principal la migración en la tecnología del componente para la administración de licenciamiento y colaboración de AUDAT, de forma tal que contribuya a la eficiencia y eficacia de este proceso. Para el cumplimiento de este objetivo se realiza un estudio preliminar sobre las soluciones existentes relacionadas con los procesos de licenciamiento y colaboración de software. Para guiar la propuesta de solución se empleó como metodología de desarrollo de software Proceso Unificado Ágil en su variación para el proceso productivo de la Universidad de las Ciencias Informáticas. Además, se utilizaron las herramientas, lenguajes y tecnologías teniendo en cuenta las características del presente trabajo. Al mismo tiempo, para garantizar la calidad del sistema se realizaron las validaciones correspondientes y se aplicaron las pruebas pertinentes.

PALABRAS CLAVE: colaboración, componente, contrato, licenciamiento, migración.

SUMMARY

The University of Computer Sciences (UCI) maintains a relationship through a macro contract with the General Comptroller's Office of the Republic of Cuba, where specialists from the CREAD center, located at UCI, work on different versions of the Data Audit System (AUDAT). The audit process revealed several inconsistencies, including the use of outdated technology and the inability to share documents for better interaction and performance in the task. Therefore, the main objective of this diploma work is to migrate the technology of the component for the licensing and collaboration management of AUDAT, in a way that contributes to the efficiency and effectiveness of this process. To achieve this objective, a preliminary study is conducted on existing solutions related to software licensing and collaboration processes. The Agile Unified Process methodology was employed to guide the proposed solution, adapted to the productive process of the University of Computer Sciences. Furthermore, tools, languages, and technologies were used, taking into account the characteristics of

this work. At the same time, to ensure the quality of the system, corresponding validations were performed, and relevant tests were applied.

KEYWORDS: *collaboration, component, contract, licensing, migration.*

Índice

TABLA DE CONTENIDOS

Índice

ÍNDICE DE TABLAS

ÍNDICE DE ILUSTRACIONES

INTRODUCCIÓN

El desarrollo vertiginoso de la auditoría automatizada ha sido un tema de interés en la industria auditora y de contabilidad en todo el mundo. La automatización de este proceso está encaminado a mejorar su eficiencia y calidad, permitiendo a los auditores centrarse en tareas de mayor valor añadido.

En los últimos años, con el aumento de la adopción de tecnologías de automatización en la auditoría y la utilización de inteligencias artificiales, aprendizajes automáticos y robóticas, los reguladores y organismos de supervisión de todo el mundo están prestando cada vez más atención a la auditoría automatizada y los auditores realizan su trabajo de forma más competente y concisa, además de, analizar grandes cantidades de datos con mayor celeridad y exactitud, mejorando la calidad del proceso (1)

En algunos países, se desarrollan regulaciones específicas para la auditoría automatizada. Se establecen estándares para asegurar que la auditoría automatizada cumpla con los mismos requisitos de calidad y éticos que la auditoría tradicional.

En cuanto a las empresas punteras en el desarrollo de la auditoría automatizada en el mundo, existen varias compañías que están liderando el camino en este campo. Algunas de las más destacadas incluyen a KPMG International, Deloitte, PricewaterhouseCoopers (PwC) y Ernst & Young (EY). Además, varias empresas de tecnología han desarrollado soluciones de automatización de auditoría, como ACLAnalytics, CaseWareIDEA y TeamMate (2).

En el caso particular de Cuba, este proceso de automatización se encuentra en una fase incipiente. Centrándose principalmente en la implementación de herramientas informáticas para la gestión de los datos financieros de las empresas (identificando posibles irregularidades y fraudes en estos) y herramientas de análisis de datos (diseñadas para analizar grandes cantidades de datos y detectar patrones y relaciones ocultas que pueden ser indicativos de actividades fraudulentas).

Se desarrollan también sistemas de contabilidad y gestión que permiten a los auditores acceder a la información de estas empresas concretamente. Sin embargo, el uso de las más avanzadas tecnologías de automatización (mencionadas anteriormente) aún no es muy común en nuestro país, a pesar de que se han realizado esfuerzos para implementarlas, se enfrentan a desafíos en términos de capacitación de los auditores y la falta de infraestructura tecnológica adecuada (3).

La Contraloría General de la República (CGR) es el organismo encargado de llevar a cabo auditorías a empresas y entidades en Cuba, funciona de manera autónoma, sin estar subordinada a ningún otro organismo del Estado. Esta institución fue creada en el año 2009 durante la Asamblea Nacional del Poder Popular, la cual desempeña un papel fundamental en la supervisión de la gestión económica y financiera de las empresas y entidades del país, promueve la cultura del control y la fiscalización, con la finalidad de garantizar la transparencia y eficiencia en la gestión pública, siendo imparciales y objetiva, mediante la implementación de sistemas de control interno y la capacitación del personal encargado de llevar a cabo estas actividades (4).

La Ley 107/2009 "De la Contraloría General de la República de Cuba" fue aprobada en el Tercer Período Ordinario de Sesiones de la VII Legislatura de la Asamblea Nacional del Poder Popular de Cuba, el 1 de agosto de 2009. Esta ley es la norma jurídica principal que regula y establece las bases legales para el funcionamiento de la Contraloría General de la República de Cuba, definiendo su estructura, funciones y atribuciones, y estableciendo las obligaciones y responsabilidades de los órganos y entidades del Estado en relación con su labor de control y fiscalización (5).

En su estrategia de informatización la CGR, se vincula con Organismos de la Administración Central del Estado (OACEs), empresas de servicios y universidades. En este último grupo, se encuentra la Universidad de las Ciencias Informáticas (UCI), con la cual mantiene relación a través del Contrato Macro o Acuerdo de Colaboración UCI-CGR, el cual ha permitido con especialistas del Centro DE Representación y Análisis de datos (CREAD), el desarrollo y evolución de las diferentes versiones del producto de software Sistema de Auditoría de Datos (AUDAT): sistema de auditoría utilizado por la CGR para la gestión y seguimiento de las auditorías realizadas en el país (6).

El sistema AUDAT está diseñado para mejorar la eficacia y la eficiencia de los procesos de auditoría en Cuba, permitiendo una gestión más eficiente y efectiva de los recursos y proporcionando información valiosa para la toma de decisiones y la mejora continua de los procesos de auditoría, siendo la versión 2.0 la última versión estable que se encuentra en uso por los clientes y actualmente se encuentra en soporte. AUDAT 2.0 permite realizar la auditoría a los sistemas de gestión que están instalados en las entidades auditadas, principalmente los sistemas contables financieros (7).

Las funcionalidades son visualizadas por el sistema a través del menú principal Archivo y los submenús Tabla, Datos, Análisis y Muestreo. Brinda la posibilidad de filtrar registros con el uso del editor de ecuaciones. Permite detectar duplicados de facturación y las omisiones en comprobantes de opera-

ciones. Sirve de apoyo al trabajo del auditor ya que los tipos de análisis y muestreos que soporta pueden ser exportados a papeles de trabajo atendiendo a las normas cubanas de auditorías, lo cual contribuye a la optimización del tiempo y permite realizar análisis más detallados y profundos de los temas auditados (7).

Cuando el sistema fue entregado a los clientes cumpliendo con sus requisitos y perspectivas, su explotación por los auditores reveló que el empleo de la misma posee una serie de problemas y necesidades entre los que destacan:

- Aunque el sistema ya cuenta con un medio de administración y licenciamiento no cumple con los requerimientos necesarios actualmente, las tecnologías obsoletas representan vulnerabilidades por lo que están expensas a fallos. Este cambio permitirá un software actualizado.
- El sistema de licenciamiento de software de AUDAT no cumple con las condiciones que se necesitan en la actualidad, permitiendo que el programa sea distribuido por diferentes computadoras sin previa autorización.
- Se entorpece el proceso de auditoría al no ser posible compartir información entre auditores.

El análisis de estas dificultades permitió identificar que AUDAT cumple parcialmente con las necesidades del cliente y que persisten insatisfacciones en dicho proceso. Se puede afirmar que AUDAT 2.0 carece de una licencia de Administración para el licenciamiento y colaboración del mismo.

A partir de la situación anteriormente descrita y los elementos expuestos, surge como **problema de la investigación** ¿Cómo contribuir a la mejora del componente para la administración de licenciamiento y colaboración de AUDAT?

La investigación tiene como **objeto de estudio**: métodos de licenciamiento de software, enmarcado en el **campo de acción:** Administración de licenciamiento y colaboración para sistemas de auditoría automatizada.

Para darle solución al problema planteado se define como **objetivo general de la investigación:** desarrollar el componente para la administración de licenciamiento y colaboración de AUDAT.

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

- Sistematización de los referentes teóricos asociados al desarrollo de sistemas informáticos para la administración del licenciamiento y colaboración de AUDAT.
- Análisis de soluciones existentes asociadas a la administración de licenciamiento de software.
- Definición de las herramientas, tecnologías y metodología a utilizar para el desarrollo del componente para la administración de licenciamiento y colaboración de AUDAT.
- Realización del análisis y diseño para el desarrollo del componente para la administración de licenciamiento y colaboración de AUDAT.
- Implementación del componente para la administración de licenciamiento y colaboración de AUDAT.
- Aplicación de las pruebas para la verificación del correcto funcionamiento del componente para la administración de licenciamiento y colaboración de AUDAT.

Para el desarrollo de la investigación se utilizarán los métodos científicos siguientes:

Métodos teóricos:

Histórico-Lógico: Para el presente trabajo se realiza un estudio del estado del arte de la problemática en cuanto a características, funcionalidades que soportan, ventajas y desventajas; así como las herramientas y tecnologías necesarias para implementar el componente para la administración de licenciamiento y colaboración de AUDAT.

Analítico-Sintético: Se utiliza para el análisis de documentos, materiales y temas relacionados con el desarrollo del componente para la administración del licenciamiento y colaboración de AUDAT. Permite el estudio de los conceptos asociados al tema. La síntesis permite realizar un análisis del impacto de los resultados del sistema a desarrollar como parte de la validación y cumplimiento de los objetivos de la presente investigación.

Modelación: Este método se utilizó para la realización de los artefactos correspondientes al análisis, diseño e implementación de las funcionalidades del sistema.

Métodos empíricos:

Entrevistas semi-estructuradas: Las entrevistas se aplican a los especialistas del área para obtener la información necesaria en el desarrollo del componente para la administración de licenciamiento.

Análisis documental: El análisis documental se realizará para consolidar los conocimientos asociados a los sistemas de licenciamiento, funciones y otros temas presentes en la investigación.

Recolección de información: Se utilizó mediante la aplicación de técnicas de recolección de información como entrevistas no estructuradas y análisis de documentos. Combinación que permitió la recogida de información durante el proceso de ingeniería de requisitos y la consulta de materiales y normativas para la realización del presente trabajo.

El siguiente trabajo de diploma consta de tres capítulos:

Capítulo 1: Fundamentación teórica-metodológica

En este capítulo se describen los aspectos teóricos relacionados con el licenciamiento y colaboración de AUDAT, se utilizarán como base para la investigación propuesta. Se proponen las herramientas y tecnologías que se deben utilizar para la implementación del componente para la administración de licenciamiento y colaboración de AUDAT. Además, se describen los patrones de diseño y arquitectónico a tener en cuenta para la implementación de la propuesta de solución a utilizar en el sistema AUDAT para dar solución a la situación problemática planteada.

Capítulo 2: Análisis y diseño del componente para la administración de licenciamiento y colaboración de AUDAT.

Este capítulo comprende las etapas de análisis y diseño de la solución propuesta. Se muestra el modelo del dominio para facilitar la comprensión del contexto del problema. Se definen las principales funcionalidades a desarrollar mediante historias de usuario. También se presenta la metodología AUP en su variación para la UCI, específicamente en su escenario 4 y los artefactos generados en la etapa de diseño.

Capítulo 3: Implementación y Validación de la solución propuesta

Este capítulo se enfoca en la fase de codificación del desarrollo del software, donde se lleva a cabo la implementación de cada funcionalidad y se aplican estándares de codificación a utilizar. Se realizan las estrategias de pruebas definidas para el sistema y se muestran los resultados de las mismas.

Por último, se presentan las conclusiones, la bibliografía y el conjunto de anexos para una mejor comprensión de lo expuesto en la investigación.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA-METODOLÓGICA

El presente capítulo recoge los elementos teóricos que fundamentan el desarrollo de un componente para la administración de licenciamiento y colaboración del sistema informático de AUDAT. Seguidamente, se reflejan las características esenciales de las diferentes soluciones informáticas existentes en el mercado para el licenciamiento de software. Finalmente, se exponen los elementos más relevantes de la metodología, lenguajes y herramientas definidas para el desarrollo del mismo.

1.1 Conceptos asociados

Para un mejor entendimiento de la propuesta, luego de una consulta de la documentación relacionada con el objeto de estudio, la autora de la presente investigación asume los siguientes conceptos

Auditoría: La auditoría es un proceso crucial para verificar el cumplimiento de las normativas y procedimientos internos dentro de las organizaciones. No obstante, en la actualidad, muchas empresas no cuentan con un documento de instrucciones que permita llevar a cabo el proceso de auditoría de manera efectiva. Por esta razón, es fundamental que los auditores se preparen adecuadamente para llevar a cabo su trabajo (8).

En este sentido, se recomienda la implementación de un plan de auditoría que incluya la definición de objetivos y alcances, la selección de herramientas y técnicas adecuadas, así como la elaboración de un cronograma de trabajo. Además, es importante que los auditores realicen una evaluación de los riesgos y oportunidades que pueden presentarse durante el proceso de auditoría, esto les permitirá identificar las áreas críticas de la organización y enfocar su trabajo en las áreas que representen un mayor riesgo para la empresa (8).

Para llevar a cabo una auditoría efectiva, los auditores deben aplicar técnicas como entrevistas con los jefes departamentales, observación directa de las actividades y procesos de ejecución, y revisión de los registros disponibles. Estas técnicas les permitirán obtener una visión completa de las actividades de la organización y asegurarse de que se cumplan las normas de calidad, seguridad y legislación gubernamental y fiscal (8).

Los auditores deben trabajar en estrecha colaboración con los jefes departamentales y otros miembros de la organización para garantizar que se aborden y solucionen los problemas identificados en el

proceso de auditoría. De esta manera, se puede mejorar continuamente el sistema productivo de la organización y garantizar su conformidad con las normativas y procedimientos internos (8)**.**

En general, se entiende que la auditoría es un estudio sistemático que se realiza a un producto, proceso o entidad y que debe documentarse para evaluar su estado actual. A partir de ahí, se realizan evaluaciones para determinar la importancia o eficiencia de lo que se está auditando: si cumple o no con lo establecido o con las condiciones prescritas (9).

Varios autores han coincidido en que la auditoría se puede clasificar en diferentes tipos, como los siguientes:

A través del personal que realiza la auditoría:

- Auditoría Interna
- Auditoría Externa

A través del objeto de análisis:

- Auditoría de Gestión
- Auditoría Operativa
- Auditoría Financiera
- Auditoría Contable
- Auditoría Informática

A través de la profundidad del análisis:

- Auditoría Total
- Auditoría Parcial

Frecuencia del análisis:

- Auditoría Continua
- Auditoría Ocasional

Interfaz Gráfica de Usuario: Es un tipo de interfaz de usuario también conocida como GUI (GraphicalUser Interface), por sus siglas en inglés. Siendo un sistema de interacción entre el usuario y un dispositivo o programa; su objetivo principal es representar el código del backend de un sistema de la

forma más clara posible para el usuario, utilizando elementos gráficos como íconos, menús e imágenes, simplificando y facilitando las tareas del usuario humano (10).Dentro de esta se encuentra la **Interfaz Colaborativa** la cual es un entorno en el que varias personas pueden trabajar juntas de manera simultánea y colaborativa en un proyecto, permitiéndoles compartir y editar documentos, archivos e imágenes de forma conjunta y en tiempo real (11).

Protección de software: Se refiere a las medidas y técnicas utilizadas para salvaguardar el código fuente y los programas de software contra accesos no autorizados, copias ilegales, modificaciones no autorizadas u otros tipos (12). Es de gran importancia para la revolución informática puesto que radica en varios aspectos que de no cumplirse traen consigo negatividades, como lo son:

- La protección de la propiedad intelectual: Pudiera generar pérdidas financieras significativas para los desarrolladores y propietarios del software.
- Mantenimiento de la confianza del cliente: Puede tener impacto negativo en la reputación e imagen de una empresa.
- Evitar el malware y las vulnerabilidades: Estas podrían ser explotadas por los atacantes para acceder a datos confidenciales o causar daños.
- Cumplimiento legal: Siendo exigida a través de leyes y regulaciones en muchos países es fundamental para evitar sanciones legales y daños de reputación.

La elección de la protección adecuada dependerá del nivel de seguridad requerido y de las necesidades específicas del software y sus usuarios, es por esto, que relacionado a la investigación es de gran importancia indagar acerca de los **Métodos de licenciamiento** (estos últimos no son más que las medidas a utilizar para prevenir la distribución o uso no autorizado de un software, su objetivo principal es proteger los derechos de autor y los intereses comerciales del propietario del software) (13), son los requerimientos de seguridad pertinentes. A continuación se muestra ejemplos de estos:

- **Criptografía:** Es la ciencia de representar información de forma opaca para que sólo los agentes autorizados (personas o dispositivos diversos) sean capaces de desvelar el mensaje oculto. Este proceso es conocido como cifrado y descifrado (14). Además, que la **Criptografía de Cifrado** como técnica utilizada para convertir información legible en un formato ilegible (texto cifrado), tiene como objetivo principal proteger la confidencialidad de los datos, teniendo como

base un conjunto de reglas y operaciones matemáticas que se utilizan para transformar estos datos en formato ilegible, para luego revertir el proceso (15).

- **Marcas de agua digitales:** Técnicas que se utilizan para identificar y rastrear el uso no autorizado del software. Estas marcas se pueden incrustar en el código fuente o en los archivos ejecutables del software (16).
- **Protección de hardware**: Se utiliza para vincular el software a un dispositivo específico, como una llave USB o un disco duro. Esto ayuda a proteger el software contra la copia no autorizada y la distribución ilegal (17).
- **Registro de derechos de autor:** Proceso legal donde se le otorga al propietario del software el derecho exclusivo de reproducir, distribuir y modificar el software. Este proceso ayuda a proteger el software contra la infracción de derechos de autor (18).
- **Licenciamiento de Software:** Son contratos que se establecen entre los desarrolladores o vendedores de software y los usuarios finales para proteger los derechos de propiedad intelectual de estos, además de evitar el uso y la distribución no autorizada de sus productos (19).

  En el caso de Cuba, la Ley No. 88 "Ley de la Propiedad Industrial" establece las normas y regulaciones en relación con la propiedad intelectual, incluyendo los derechos de autor y el licenciamiento de software (20). Además, la Oficina Cubana de la Propiedad Industrial (OCPI) es el organismo encargado de la protección y gestión de los derechos de propiedad industrial en Cuba, incluyendo los derechos de autor sobre el software (21).
  Las licencias de software pueden incluir una serie de cláusulas y restricciones que el usuario final debe cumplir para poder utilizar el software. Estas cláusulas y restricciones pueden variar según los Métodos de licenciamiento de software que se esté utilizando.

Software de colaboración: Herramienta informática diseñada para facilitar la colaboración y el trabajo conjunto entre usuarios. Este tipo de software permite compartir información, documentos, tareas, calendarios, mensajes y otros recursos de manera eficiente, lo que ayuda a mejorar la productividad y la comunicación dentro de los equipos de trabajo. Persigue el desarrollo del conocimiento compartido, la aceleración de los flujos de información, la coordinación de los flujos de recursos para reducir economías de costos y tiempo (22).

Software para el Sistema de Auditoría de Datos (AUDAT) : El Sistema de Auditoría de Datos (AUDAT) es un software desarrollado por la Universidad de Ciencias Informáticas (UCI) de Cuba, en co-

laboración con la Contraloría General de la República de Cuba (CGR), para la gestión y seguimiento de las auditorías realizadas por la CGR a las empresas y entidades en Cuba, es una herramienta informática que permite la automatización de los procesos de auditoría, desde la planificación hasta la emisión del informe final de la auditoría. El software incluye módulos para la gestión de las diferentes etapas del proceso de auditoría, como la planificación, la ejecución, el análisis de resultados y la elaboración del informe final, así como para la gestión de la documentación y la comunicación con las empresas y entidades auditadas.

Tiene como objetivo principal optimizar el tiempo y los recursos utilizados en las auditorías, mejorar la calidad de los informes de auditoría y facilitar la comunicación y la colaboración entre la CGR y las empresas y entidades auditadas. También contribuye a la mejora continua de los procesos de auditoría y al fortalecimiento del control interno en las empresas y entidades en Cuba (7).

A raíz del análisis con el equipo de desarrollo del software para el Sistema de Auditoría de Datos (AUDAT), se arriba a la siguiente conclusión: El sistema ya está implementado, pero presenta deficiencias, como son la colaboración y protección del producto, es recomendable utilizar una Licencia Comercial, puesto que, para ello el usuario solo puede usar la interfaz del software pagando una tarifa por la copia de la licencia. No hay acceso al código fuente de la aplicación, estas protegen los intereses de los desarrolladores en los activos de sus aplicaciones. Los desarrolladores crean la clave de licencia o el código para que los usuarios no puedan instalar y activar la herramienta en más de una computadora si la clave es para un usuario, o según el acuerdo, encriptando así el software.

1.2 Análisis de soluciones existentes

Luego del análisis de los principales conceptos asociados al dominio del problema, se realiza un estudio de los sistemas con el objetivo de definir sus características y la viabilidad de su uso en la solución de la problemática.

Existen varios softwares, que pueden ayudar a encriptar los procesos auditores, ofreciendo así funciones de colaboración y medidas de seguridad para proteger la integridad y confidencialidad de los datos, algunos de los cuales son:

Citrix ServiceProvider: El programa proporciona soluciones de virtualización bajo un régimen de suscripción o alquiler. Los proveedores de servicios podrán ofrecer a sus clientes productos de virtualización de Citrix, incluyendo equipos de escritorio y aplicaciones hospedadas, a través de suscripción o

alquiler, sin compromisos iniciales o cuotas mínimas, pagando mensualmente tan solo por lo utilizado. Cuenta con el mejor rendimiento, la mayor seguridad y el menor costo del mercado. Ofrece flexibilidad y simplicidad, sin compromisos de pago por adelantado de la licencia. Este programa ayuda a sacar el máximo provecho de los últimos modelos de negocio basados en la nube (22).

Microsoft SPLA: Programa de licenciamiento mensual que permite arrendar licencias de Microsoft, para brindar servicios de software y aplicaciones hosted a sus clientes. Se trata de un programa de licencias que permite a los proveedores de servicios y vendedores independientes de software (ISVs: por su acrónimo en inglés), la posibilidad de alquilar licencias mensualmente para prestar servicios y aplicaciones de hosting a clientes finales, sin la necesidad de realizar una inversión de capital inicial y evitando los riesgos correspondientes. Posibilita que los proveedores de servicios puedan ofrecer alojamiento, externalización y otros servicios, evitando costos iniciales, planificando el presupuesto de licencias y pagando sólo por lo que se utiliza (23).

Symantec ExSP Program: Programa de licenciamiento para Proveedores de Servicios que permite la adquisición de las licencias bajo un modelo de pago por uso (pay-per-use) a través de una suscripción mensual y predecible. Este programa brinda a los partners la flexibilidad que necesitan para combinar software de Symantec con sus servicios, permitiéndoles adaptar mejor el uso del software con sus programas de pagos. Las ofertas de productos de Symantec permiten a los partners brindar a sus clientes soluciones de seguridad y de disponibilidad. Pueden agregar o eliminar productos para satisfacer las complejas necesidades comerciales de sus clientes (24).

VMware Cloud ProviderProgram: Permite a los socios invertir en la infraestructura de VMware sobre una base de suscripción mensual, impulsando el éxito de las nubes públicas e híbridas. El Programa se compone de las ofertas de software como servicio de VMware y nuestro ecosistema global de VMwareServiceProviderPartners. Es la solución ideal para todas las empresas que ofrecen servicios alojados a terceros, incluidos proveedores de infraestructura como servicio (IaaS), proveedores de servicios en la nube (CSP), proveedores de servicios de aplicaciones (ASP), proveedores de servicios de Internet (ISP) servicio (PaaS) (25).

Software	Tipo de Licencia	Tipo de Código	Algoritmo de Cifrado	Colaboración de Software	Interacción
Citrix Service-Provider	Privativa	Privado	Asimétrico	Ostenta	Tiene
Microsoft SPLA	Privativa	Privado	Asimétrico	Ostenta	Tiene
Symantec ExSP Program	Privativa	Privado	Asimétrico	Ostenta	Tiene
VMware Cloud Provider Program	Privativa	Privado	Asimétrico	Ostenta	Tiene

Tabla 1 Análisis de soluciones existentes.

Fuente: Elaboración propia.

Valoración sobre el estudio de soluciones existentes

Luego de realizado el análisis de las soluciones existentes descritas anteriormente, se puede plantear que en su generalidad no satisfacen la necesidad explicada en la problemática. Existe una similitud en cuanto a la utilización del algoritmo de cifrado simétrico (se utiliza una clave única para cifrar y descifrar los datos, es una clave previamente acordada entre el remitente y el destinatario). Lo contrario sucede a razón del tipo de licencia y de código con el que cuentan estos softwares, pues no será posible hacer uso ni del programa en general, ni del código en lo particular ya que las empresas creadoras de estos mantienen la propiedad intelectual de su código fuente. Otra pauta es la colaboración de software y la interacción de los usuarios en el sistema dado que los softwares comparados cuentan con estas y AUDAT no dispone de estas características.

1.3 Metodología de Desarrollo

En la actualidad, la industria del software ha evolucionado rápidamente, lo que ha llevado a replantear los enfoques tradicionales de desarrollo de software, existe una gran variedad de metodologías para el desarrollo de software, estas se pueden clasificar en dos grandes grupos: las metodologías tradicionales o pesadas y las metodologías ágiles (26).

Las metodologías tradicionales se basan en las buenas prácticas de la ingeniería del software y se enfocan en seguir un riguroso proceso de aplicación y un marco disciplinado. Estas metodologías se aplican en proyectos que requieren planificación detallada y documentación exhaustiva de cada fase del proceso de desarrollo. Este enfoque es adecuado para proyectos de gran escala y complejidad, donde se requiere un alto nivel de control y gestión de riesgos (26).

Por otro lado, las metodologías ágiles representan una solución a los problemas que requieren una respuesta rápida en un ambiente flexible y con cambios constantes, se enfocan en la entrega continua de software funcional, haciendo un uso menos estricto de documentación y métodos formales. Este enfoque es adecuado para proyectos que requieren una adaptabilidad constante a los cambios en los requerimientos del proyecto y una respuesta rápida a las necesidades del cliente (26).

Metodología AUP: La metodología AUP es una metodología de desarrollo de software creada por Scott Ambler, que se considera una versión simplificada del Proceso Unificado Racional (RUP), se enfoca en la entrega iterativa y en la colaboración constante entre el equipo de desarrollo y el cliente, lo que se considera una práctica ágil. Además, incluye técnicas ágiles como el Desarrollo Dirigido por Pruebas, el Modelado Ágil, la Gestión de Cambios y la Refactorización de Base de Datos para mejorar la productividad.

Para adaptar la metodología AUP a las necesidades específicas de cada proyecto, se puede considerar la inclusión de nuevas técnicas ágiles, como el Desarrollo de Software Basado en Componentes, el Desarrollo de Software Basado en Prototipos o el Desarrollo de Software Basado en Pruebas Continuas. Estas técnicas pueden ayudar a mejorar la calidad del software producido y acelerar el tiempo de entrega (27).

Su estructura se organiza en cuatro fases principales: Inicio, Elaboración, Construcción y Transición. Cada fase se enfoca en una etapa específica del ciclo de vida del software y en la entrega iterativa del software funcional (27).

Las disciplinas que se definen en la metodología AUP se dividen en dos categorías: procesos ingenieriles y gestión de proyectos. Los procesos ingenieriles incluyen cuatro disciplinas:

- Modelado (comprensión de los requisitos del cliente y la creación de modelos que describan el sistema a desarrollar).
- Implementación (codificación y desarrollo del software).

- Pruebas (la validación del software y la disciplina).
- Despliegue (entrega del software a los usuarios finales).

Las disciplinas de gestión de proyectos incluyen tres disciplinas:

- Gestión de Configuración y Cambios (la gestión de versiones del software y en el control de cambios).
- Gestión de Proyectos (planificación y gestión del proyecto).
- Ambiente (configuración del ambiente de desarrollo, pruebas y producción).

La variación propuesta de la metodología AUP para los proyectos de la UCI incluye ocho disciplinas, con un enfoque más atómico que el definido en AUP. Se incluyen disciplinas adicionales para cubrir todas las etapas del ciclo de vida del software y se enfoca en la entrega de software de alta calidad. Además, se alinea con el modelo CMMI-DEV v1.3 para asegurar la aplicación de las mejores prácticas en el desarrollo de software.

Cuenta además con 4 escenarios, de los cuales se dará una breve explicación:

Escenario No 1: Aplica a los proyectos que hayan evaluado el negocio a informatizar y como resultado obtengan que puedan modelar una serie de interacciones entre los trabajadores del negocio/actores del sistema (usuario), similar a una llamada y respuesta respectivamente, donde la atención se centra en cómo el usuario va a utilizar el sistema.

Escenario No 2: Aplica a los proyectos que hayan evaluado el negocio a informatizar y como resultado obtengan que no es necesario incluir las responsabilidades de las personas que ejecutan las actividades, de esta forma modelarían exclusivamente los conceptos fundamentales del negocio.

Escenario No 3: Aplica a los proyectos que hayan evaluado el negocio a informatizar y como resultado obtengan un negocio con procesos muy complejos, independientes de las personas que los manejan y ejecutan, proporcionando objetividad, solidez, y su continuidad.

Escenario No 4: Aplica a los proyectos que hayan evaluado el negocio a informatizar y como resultado obtengan un negocio muy bien definido. El cliente estará siempre acompañando al equipo de desarrollo para convenir los detalles de los requisitos y así poder implementarlos, probarlos y validarlos (28).

En la metodología seleccionada existen 4 escenarios para la disciplina de Requisitos, el escenario 4 es el utilizado en la presente propuesta de solución ya que aplica a proyectos que obtengan un negocio bien definido. El cliente estará acompañando al equipo de desarrollo para convenir los detalles de los requisitos encontrados y poder implementarlos, probarlos y validarlos. Se recomienda en proyectos no extensos, ya que las Historias de usuarios (HU) no deben poseer demasiada información.

1.4 Tecnologías lenguajes y herramientas

Actualmente existe una amplia variedad de tecnologías, lenguajes y herramientas que permiten la creación y el desarrollo de aplicaciones para la web. Se decide utilizar las siguientes tecnologías ya que están en consonancia con la estrategia de soberanía tecnológica por la que opta el país. A continuación, se describen las tecnologías lenguajes y herramientas que conforman el ambiente de desarrollo del componente para la administración del licenciamiento y colaboración.

1.4.1 Herramienta y lenguaje de modelado

Visual Paradigm: es una herramienta CASE4 que utiliza UML como lenguaje del modelado. Esta herramienta está pensada para usuarios interesados en construir sistemas de software fiables con el uso del paradigma orientado a objetos, incluyendo actividades en las que se destacan: Ingeniería de Software, análisis de sistemas y análisis de negocio. La utilización de la herramienta en cuestión ayudará a aumentar la productividad durante el desarrollo de la solución, automatizando parte del proceso, además, la documentación estará enriquecida con diagramas para lograr un mejor entendimiento del funcionamiento del sistema para futuros desarrolladores (29).

Se decide trabajar con el Visual Paradigm (v8.0) para UML por su robustez, usabilidad y portabilidad. Además, se cuenta con la licencia para su utilización y soporta el ciclo completo del proceso de desarrollo de software.

UML: es un lenguaje basado en diagramas para la especificación, visualización, construcción y documentación de cualquier sistema complejo, se basa en el paradigma orientado a objeto, es un modelo que simplifica la realidad que construimos para comprender mejor el sistema a desarrollar; proporciona los planos de un sistema, incluyendo tanto los que ofrecen una visión global del sistema como lo más detallado de alguna de sus partes (30).

Se selecciona el lenguaje de modelado UML (v2.0) para la confección de varios productos de trabajo de la solución. Se decide la utilización de este lenguaje de modelado, pues posibilita estandarizar la

documentación referente al proceso de desarrollo de software; es uno de los lenguajes de modelados más utilizados y difundidos.

1.4.2 Lenguaje de programación

Los lenguajes de programación se utilizan para definir una secuencia de instrucciones para su procesamiento por un ordenador o computadora. Se asume generalmente que la traducción de las instrucciones a un código que comprende la computadora debe ser completamente sistemática. Estos se componen de un conjunto de reglas sintácticas y semánticas que permiten expresar instrucciones que luego serán interpretadas (31). En este epígrafe se hace referencia sobre diversos lenguajes de programación empleados en la confección de la propuesta de solución. Estos son utilizados como un sistema para dar forma a la plataforma web y obtener una aplicación funcional que resuelva la problemática de esta investigación.

Python: es un lenguaje fácil de aprender. Posee estructuras de datos eficientes y de alto nivel y un enfoque simple pero efectivo a la programación orientada a objetos. Tiene una cómoda sintaxis y un tipado dinámico, junto con su naturaleza interpretada, hacen de éste un lenguaje ideal para desarrollo rápido de aplicaciones en diversas áreas y sobre la mayoría de las plataformas (32).

Se decidió emplear el lenguaje de programación Python (v3.8), ya que es un lenguaje que hace referencia a su propia limpieza y legibilidad, está desarrollado bajo una licencia de código abierto, por lo que es de libre uso y distribución, incluso para uso comercial. Posee una gran comunidad a nivel mundial que están constantemente aportando nuevas mejoras. Posee buen rendimiento, además su característica multiplataforma permite que este pueda ser usado en diferentes sistemas operativos.

1.4.3 Bibliotecas y marcos de trabajo para el desarrollo

Django: es un marco de trabajo (framework) para el desarrollo de aplicaciones web usando Python. Django considera algunas funcionalidades listas para usar facilitando el desarrollo de aplicaciones web. Como resultado, no es necesario escribir todo el código ni usar tiempo para buscar errores de código en el framework. Es decir, mediante Django, el desarrollo de sistemas de información web puede ser rápido, seguro, escalable y también fáciles de mantener (33).

Se decidió emplear el framework Django (v4.0) debido a que permite construir y mantener aplicaciones web de alta calidad con el mínimo esfuerzo posible. Igualmente es un potente marco de trabajo que posee una arquitectura bien definida. Posee varios módulos integrados para la seguridad y poder

realizar operaciones básicas con solo unas pocas líneas de códigos, incrementando la calidad de las soluciones, aumentando la productividad y disminuyendo los errores en el código.

1.4.4 Sistema gestor de base de datos

Un Sistema Gestor de Bases de Datos (SGBD) es un conjunto de programas que administran y gestionan la información que contiene una base de datos. Además, hacen posible administrar todo acceso a la base de datos ya que tienen el objetivo de servir de interfaz entre ésta, el usuario y las aplicaciones (34).

PostgreSQL: es un potente gestor de base de datos, bajo la licencia BSD6. Cuenta con más de 15 años de desarrollo activo y arquitectura probada que ha ganado mucha reputación por su confiabilidad e integridad en los datos. Está diseñado para ambientes de altos volúmenes de datos y al ser multiplataforma está disponible para muchos sistemas operativos (35).

En términos de licencia y recursos, es capaz de ajustarse al número de procesadores y a la cantidad de memoria que posee el sistema de forma óptima, lo que posibilita atender un mayor número de peticiones concurrentes. Se selecciona PostgreSQL (v14.2) para el desarrollo de esta aplicación por las ventajas que posee ante el desarrollo de este sistema. Es multiplataforma, tiene soporte nativo para PHP. También permite todo tipo de consultas a bases de datos. Brinda soporte multiusuario, para el control de concurrencia, dispone de los tipos de datos del estándar SQL, así como soporte para tipos de datos creados por el usuario. Además está en correspondencia, según (36), con las políticas de migración de software de la UCI.

1.4.5 Herramientas de Prueba

Las herramientas para la automatización de pruebas de software ayudan a ejecutar pruebas funcionales y de regresión en la aplicación. Estas herramientas deben producir resultados coherentes con los datos de entrada proporcionados. Existen muchas herramientas de comprobación de software disponibles en el mercado, cada una con sus propias ventajas y características (37).

Apache JMeter (versión 5.6.2): Se decide escoger esta herramienta para la realización de las pruebas automatizadas puesto que es una herramienta ideal para realizar pruebas de rendimiento en recursos estáticos o dinámicos y en aplicaciones web. Su funcionalidad se basa en la simulación de grupos de usuarios que envían solicitudes a un servidor o red y luego devuelven estadísticas a un usuario a través de diagramas visuales (38).

1.5 Conclusiones del Capítulo

El estudio de la bibliografía y análisis de las soluciones informáticas existentes para el proceso de licenciamiento y colaboración de AUDAT, suscitó una elevada comprensión del estado del arte en el área del conocimiento en cuestión, aportando los elementos teóricos que fundamentan la investigación. Por lo que al finalizar el presente capítulo se arribó a las siguientes conclusiones parciales:

- Se enunciaron los principales términos asociados al dominio de la investigación, lo que permitió adquirir un mayor conocimiento sobre el objeto de estudio de la investigación.

- Las soluciones existentes, no se adaptan a la dinámica del proceso de licenciamiento y colaboración.

- Inexistencia de licencias privativas en el software de AUDAT.

- La metodología de software que guía el proceso de desarrollo es AUP en su variación para la UCI, específicamente en el escenario 4, dada la interacción con el cliente y el requerimiento de una tecnología ágil en el desarrollo del mismo.

- Realizada una revisión de los principales elementos teóricos para el desarrollo del trabajo, se decide desarrollar un componente que permitirá, entre otras funcionalidades, exportar licencias a los usuarios.

- El lenguaje de programación (lenguaje de programación a utilizar) es Python, haciendo uso del framework Django

CAPÍTULO 2: ANÁLISIS Y DISEÑO DEL COMPONENTE PARA LA ADMINISTRACIÓN DE LICENCIAMIENTO Y COLABORACIÓN DE AUDAT.

En el presente capítulo se abordan los principales elementos relacionados con la propuesta de solución. Se identifican los requisitos funcionales y no funcionales y se encapsulan los requisitos mediante las historias de usuarios. De igual forma se describe el patrón arquitectónico que tendrá la propuesta de solución, así como particularidades del diseño e implementación. A partir de los requisitos identificados se modelan los diagramas de clases del diseño los cuales permiten obtener una visión más clara de la implementación del sistema. Además, se obtiene el diagrama de componentes que representa la relación y las dependencias entre los mismos. Por otra parte, se describen los patrones de diseños empleados y el modelo de datos de la solución.

2.1 Propuesta de solución

Una vez el usuario solicite un contrato para obtener la licencia de acceso al sistema de Auditoría de Datos, especificando la cantidad de computadoras a las que se les otorgará los permisos, este puede ser de único pago, pero se incita los clientes a comprar actualizaciones anuales y de soporte técnico por separado.

Realizada la acción anterior, el usuario se autentica al sistema con su clave única para la licencia. AUDAT verificará si está aprobada la licencia, si no existe o si está o no caducada. En el caso de estar caducada o no existir, el usuario deberá realizar el accionar del contrato de otorgamiento de licencias. En caso contrario ya podrá acceder a toda la información que contiene AUDAT y dentro de las acciones que podrá realizar se encuentran: exportar e importar documentos.

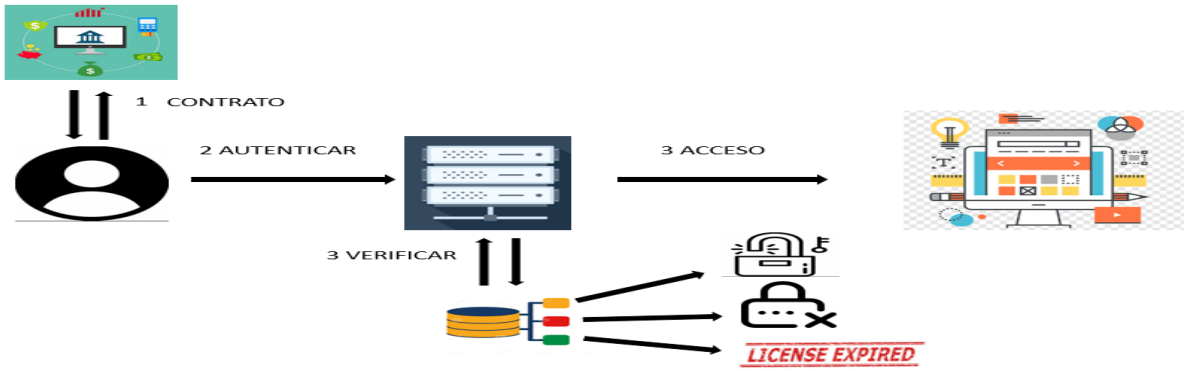

Ilustración 1 Propuesta de Solución.

Fuente: Elaboración Propia.

2.2 Modelo de dominio

Un modelo del dominio muestra clases conceptuales significativas en un dominio del problema; es el artefacto más importante que se crea durante el análisis. También se les denomina modelos conceptuales, modelo de objetos del dominio y modelos de objetos de análisis. Utilizando la notación UML, un modelo del dominio se representa con un conjunto de diagramas de clases en los que no se define ninguna operación (40). Pueden mostrar:

- Objetos del dominio o clases conceptuales.
- Asociaciones entre las clases conceptuales.
- Atributos de las clases conceptuales.

2.2.1 Descripción de los conceptos

En la siguiente tabla se describen los conceptos asociados al modelo conceptual:

Conceptos	Descripción
CGR (Contraloría General de la República)	Organismo encargado de llevar a cabo las auditorías en Cuba.
Contrato	Documento legal que se utiliza para oficializar un acuerdo entre dos o más partes.
Licencia	Autorización que otorga un autor o autores que permite el derecho de utilizar su creación o recurso.
Usuario	Persona que utiliza un sistema informático.

Tabla 2 Conceptos asociados al modelo conceptual.

Fuente: Elaboración Propia.

En la siguiente figura se representa los conceptos relevantes y sus relaciones en la descripción de la solución que se propone:

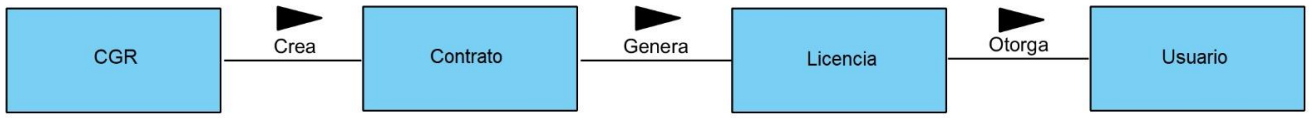

Ilustración 2 Modelo Conceptual.

Fuente: Elaboración propia.

2.2 Especificación de requisitos

Para lograr un entendimiento entre clientes y el equipo de proyecto a lo largo del proceso de desarrollo del sistema, se deben establecer los objetivos del producto a desarrollar. Con el fin de llegar a un consenso entre ambas partes, surge la especificación de requisitos (39)donde se analizan las necesidades exactas de los usuarios. Luego son traducidas a precisas funciones y acciones que serán usadas en el desarrollo del sistema. Una correcta identificación de requisitos permite posteriormente la construcción de un software de calidad. Estos pueden clasificar por categorías en: funcionales y no funcionales, además permiten comprender desde un inicio, la línea a seguir en el desarrollo y así garantizar la calidad del sistema.

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 cuando se cumplen ciertas condiciones (40). Para darle solución al problema planteado que da paso a la presente investigación, se identificaron 6 RF, en la Tabla 2 se muestra la especificación de estos requisitos:

#RF	Requisito Funcional	Descripción del Requisito Funcional	Prioridad para el cliente	Complejidad
RF 1	Solicitar licencia	El usuario solicita la licencia a partir de un contrato UCI-CGR.	Alta	Alta
RF 2	Obtener licencia	El sistema entregará al usuario su licencia	Alta	Alta
RF 3	Validar licencia	El sistema realizará una verificación de la licencia.	Alta	Alta

RF 4	Importar documentos	Dentro del sistema existe la opción de importar documentos.	Alta	Alta
RF 5	Exportar documentos	Dentro del sistema existe la opción de exportar documentos.	Alta	Alta
RF 6	Listar documentos	Dentro del sistema existe la opción de listar documentos	Alta	Alta

Tabla 3 Listado de Requisitos Funcionales del sistema.

Fuente: Elaboración Propia.

2.2.2 Requisitos no funcionales

Los Requisitos No Funcionales (RNF) son propiedades o cualidades que el producto debe tener. Son restricciones de los servicios o funciones ofrecidas por el sistema, y generalmente se aplican al sistema en su totalidad. En muchos casos los requisitos no funcionales son fundamentales en el éxito del producto. Están vinculados a requisitos funcionales, es decir, una vez que se conozca lo que el sistema debe hacer se puede determinar cómo ha de comportarse, qué cualidades debe tener o cuán rápido debe ser (41).Los RNF tienen varias clasificaciones, la utilizada sigue lo que propone la plantilla de especificación de requisitos de la metodología AUP-UCI. Los requisitos identificados son (42):

Requisito de Software

1. El sistema se desplegará en un servidor que tenga instalado: sistema gestor de bases de datos PostgreSQL (v12.2) o superior, Python (v3.8) o superior.

Requisito de Hardware

2. El sistema debe poder ejecutarse en una PC cliente con hardware con microprocesadores con Intel Pentium o superior.

3. El servidor donde se va a instalar la solución informática debe disponer de aproximadamente 8GB de RAM y 500GB de espacio de almacenamiento en disco como mínimo y un procesador Intel Core i3 a 2,4GHz en el hardware de despliegue.

Requisito de Rendimiento

4. El sistema debe de tener un tiempo de respuesta a las peticiones menor a 4 segundos.

Requisito de Seguridad

5. Debe estar preparado ante ataques *Cross Site Request Forgery* (CSRF o XSRF) para evitar peticiones a una aplicación web vulnerable.

6. El sistema debe utilizar autenticación basada en *tokens* para proteger los *endpoints* sensibles.

2.3 Historias de usuario

La metodología AUP-UCI, en su escenario 4 para la disciplina requisitos, genera las Historias de Usuario (HU), estas se utilizan para especificar los requisitos de software en la aplicación. Las HU se descomponen en tareas de programación y describen las características que el sistema debe cumplir, están escritas en un formato legible por el cliente, sin necesidad de sintaxis técnicas (43).A continuación, se muestra en las siguientes tablas varias historias de usuarios las cuales fueron generadas a partir de los requisitos funcionales y las mismas se estructuran de la siguiente forma: Número: A cada HU se le asigna un número para facilitar su identificación por parte del equipo de desarrollo.

Riesgo en Desarrollo: Grado de complejidad que le asigna el equipo de desarrollo a la HU luego de analizarla. (Alto, Medio o Bajo).

Nombre: Nombre descriptivo de la HU.

Prioridad: Grado de prioridad que le asigna el cliente a la HU en dependencia del valor en el negocio. Los valores que puede tomar son (Alta, Media o Baja).

Iteración Asignada: Número de la iteración en la cual será implementada la HU.

Tiempo estimado: Esfuerzo estimado por el equipo de desarrollo para darle cumplimiento a la HU.

Tiempo real: Plazo real que el equipo de desarrollo tiene para dar cumplimiento a la HU.

Programador: Nombre del programador encargado de desarrollar la HU.

Descripción: Descripción simple sobre lo que debe hacer la funcionalidad a la que se hace referencia.

Observaciones: Alguna acotación importante a señalar sobre la historia.

Historia de Usuario	
Número: HU-1	Riesgo en Desarrollo: N.A.
Nombre de la Historia de Usuario: Solicitar Licencia.	
Prioridad: Alta	Iteración Asignada:1
Tiempo Estimado: 1 día	Tiempo real: 1 día
Programador: María del Carmen Hernández Landestoy.	
Descripción: El usuario solicita la licencia a partir de un contrato UCI-CGR. Este proceso debe realizarse con anterioridad, en este se debe plantear quien tendrá acceso a la licencia, así como la cantidad de equipos a los que se le será otorgada también.	
Observaciones: Para poder acceder al sistema esta petición se debe hacer con anterioridad.	

Tabla 4 Historia de Usuario # 1 (Solicitar Licencia).

Fuente: Elaboración Propia.

Historia de Usuario	
Número: HU-2	Riesgo en Desarrollo: N.A.
Nombre de la Historia de Usuario: Obtener Licencia	
Prioridad: Alta	Iteración Asignada:2
Tiempo Estimado: 1 día	Tiempo real: 1 día
Programador: María del Carmen Hernández Landestoy.	
Descripción: El sistema entregará al usuario su licencia luego de ser validada por ambas partes del contrato, es decir: La Universidad de las Ciencias Informáticas y la Contraloría General de la República. Esta constará de una clave única.	
Observaciones: Se muestra al usuario el estado de su licencia.	

Tabla 5 Historia de Usuario # 2 (Obtener Licencia).

Fuente: Elaboración Propia.

Historia de Usuario	
Número: HU- 3	Riesgo en Desarrollo: N.A.
Nombre de la Historia de Usuario: Validar licencia	
Prioridad: Alta	Iteración Asignada:3
Tiempo Estimado: 1 día	Tiempo real: 1 día
Programador: María del Carmen Hernández Landestoy.	
Descripción: El sistema realizará una verificación de la licencia. Una vez que el usuario se autentica al sistema, este verificará si ya le fue otorgada la licencia, si la licencia existe y el tiempo de caducidad de esta.	
Observaciones: Las verificaciones a ejecutar serán: Comprobar la existencia de la licencia, comprobar si la licencia fue aprobada, comprobar la fecha de caducidad.	

Tabla 6 Historia de Usuario # 3 (Validar Licencia).

Fuente: Elaboración Propia.

Historia de Usuario	
Número: HU- 4	Riesgo en Desarrollo: N.A.
Nombre de la Historia de Usuario: Importar documentos	
Prioridad: Alta	Iteración Asignada:4
Tiempo Estimado: 1 día	Tiempo real: 1 día
Programador: María del Carmen Hernández Landestoy.	
Descripción: Dentro del sistema existe la opción de importar documentos. Esta acción permitirá al sistema cargar los documentos.	
Observaciones: Se puede extraer uno o varios documentos.	

Tabla 7 Historia de Usuario # 4 (Importar Documentos).

Fuente: Elaboración Propia.

Historia de Usuario	
Número: HU- 5	Riesgo en Desarrollo: N.A.
Nombre de la Historia de Usuario: Exportar documentos	
Prioridad: Alta	Iteración Asignada:5
Tiempo Estimado: 1 día	Tiempo real: 1 día
Programador: María del Carmen Hernández Landestoy.	
Descripción: Dentro del sistema existe la opción de exportar documentos. Esta acción permitirá al sistema descargar los documentos.	
Observaciones: Se puede anexar al sistema uno o varios documentos.	

Tabla 8 Historia de Usuario # 5 (Exportar Documentos).

Fuente: Elaboración Propia.

Historia de Usuario	
Número: HU- 6	Riesgo en Desarrollo: N.A.
Nombre de la Historia de Usuario: Listar documento	
Prioridad: Alta	Iteración Asignada:6
Tiempo Estimado: 1 día	Tiempo real: 1 día
Programador: María del Carmen Hernández Landestoy.	
Descripción: Dentro del sistema existe la opción de listar documentos. Esta acción permite al sistema mostrar un listado con los detalles de cada documento existente.	

Observaciones: El usuario puede realizar la petición de listar los documentos que existen en el sistema.

Tabla 9 Historia de Usuario # 6 (Listar Documentos).

Fuente: Elaboración Propia.

2.4 Arquitectura del sistema

La Arquitectura de Software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución (ISO/IEC/IEEE 42010, 2021). Alude a la estructura general del software y las formas en que la estructura proporciona una integridad conceptual para un sistema (30).

En su forma más simple, la arquitectura es la estructura u organización de los componentes del programa (módulos), la manera en que estos componentes interactúan, y la estructura de datos que utilizan los componentes. En sentido más amplio, sin embargo, los componentes pueden generalizarse para representar elementos importantes del sistema y sus interacciones. Esta no es más que un conjunto de patrones que proporcionan un marco de referencia necesario para guiar la construcción de un software, permitiendo a los programadores, analistas y todo el conjunto de desarrolladores del software compartir una misma línea de trabajo y cubrir todos los objetivos y restricciones de la aplicación. Es considerada el nivel más alto en el diseño de la arquitectura de un sistema puesto que establecen la estructura, funcionamiento e interacción entre las partes del software (30).

Un patrón es una colaboración parametrizada junto con las pautas sobre cuando utilizarlo. Un parámetro se puede sustituir por diversos valores, para producir distintas colaboraciones. Los parámetros señalan generalmente las ranuras para las clases. Cuando se instancia un patrón, sus parámetros están ligados a clases reales de un diagrama de clases o a los roles dentro de una colaboración más amplia (44).

2.4.1 Patrón arquitectónico

A continuación, se va a describir el patrón arquitectónico el cual es usado en la propuesta de solución, pero antes es necesario definir qué es un patrón arquitectónico.

Los patrones arquitectónicos son parte esencial en el desarrollo de software. Cada patrón describe un problema que ocurre una y otra vez en nuestro ambiente, y luego describe el núcleo de su solución de forma tal que es posible usarla un millón de veces sin elaborarla dos veces de la misma forma (45). Estos patrones definen la estructura general del software, indican las relaciones entre los subsistemas y los componentes del software y definen las reglas para especificar las relaciones entre los elementos (clases, paquetes, componentes, subsistemas) de la arquitectura.

Model-View-Template

Django se suele llamar un framework Modelo Vista Plantilla (MVT por sus siglas en inglés) debido a que está fuertemente influenciado por Modelo Vista Controlador (MVC) y es incluso posible argumentar que la terminología MVC es el único patrón de cambios en Django. Las tres capas básicas son el modelo, la vista, y la plantilla (46).

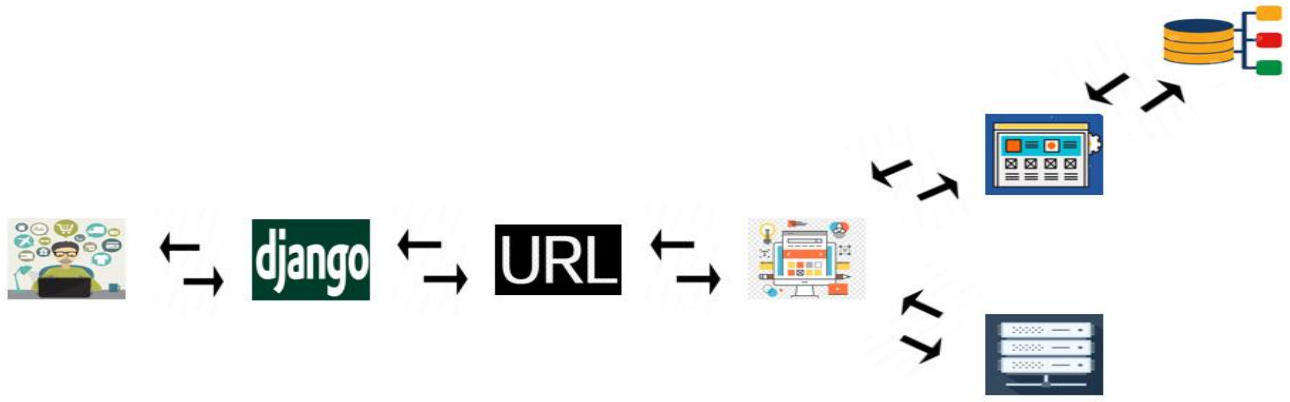

Ilustración 3 Patrón Arquitectónico (Modelo – Vista – Plantilla).

Fuente: Elaboración propia.

A continuación, se explica el funcionamiento del MVT de Django que se ejemplifica en la imagen anterior:

1. El usuario envía una petición a través del navegador web.
2. El motor de Django relaciona la solicitud con una de las urls configuradas en urls.py.
3. La url llama a la vista que le corresponde.

4. La vista interactúa con el modelo para obtener datos.

5. La vista llama a la plantilla.

6. La plantilla se renderiza en respuesta a la solicitud del navegador.

Modelo: El modelo define los datos almacenados, se encuentra en forma de clases de Python, cada tipo de dato que debe ser almacenado se encuentra en una variable con ciertos parámetros y posee métodos también. Todo esto permite indicar y controlar el comportamiento de los datos (48).

Vista: La vista se presenta en forma de funciones en Python, su propósito es determinar qué datos serán visualizados, entre otras cosas más. El ORM de Django permite escribir código Python en lugar de SQL para hacer las consultas que necesita la vista. También se encarga de tareas conocidas como el envío de correo electrónico, la autenticación con servicios externos y, la validación de datos a través de formularios.

Plantilla: Es la encargada de recibir los datos de la vista y luego los organiza para la presentación al navegador web. La plantilla es básicamente una página HTML con algunas etiquetas extras propias de Django, en sí no solamente crea contenido en HTML también XML, CSS, JavaScript, CSV, etc.

2.5 Patrones de diseño

Los patrones de diseño son soluciones simples y elegantes a problemas específicos y comunes del diseño orientado a objetos. Estos patrones hacen más fácil la reutilización del código y los diseños. Facilitan también el aprendizaje al programador inexperto facilitando la decisión entre herencia y composición. Para el diseño del componente fueron analizados patrones de gran utilidad que proponen una forma de reutilizar la experiencia de los desarrolladores clasificando y describiendo formas de solucionar problemas frecuentes en la etapa del desarrollo. Estos se evidencian en dos grupos (Patrones GRASP y Patrones GOF, por sus siglas en inglés)

2.5.1 Patrones Generales de Software para Asignación de Responsabilidades (GRASP)

General ResponsibilityAsignment (GRASP por su acrónimo en inglés) describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones. El nombre se eligió para indicar la importancia de captar estos principios, si se quiere diseñar eficazmente el software orientado a objetos. Favorecen la reutilización de código y ayudan a construir

software basado en la reutilización (49). Los utilizados en la solución son los que a continuación se muestran.

Bajo Acoplamiento: El acoplamiento es una medida de la fuerza con que una clase está conectada a otras clases, con que las conoce y con que recurre a ellas. Las URLconfs de Django son un claro ejemplo de este principio en la práctica. En una aplicación de Django, la definición de la URL y la función de vista que se llamará están débilmente acopladas.

Este patrón se evidencia dentro del marco de trabajo, en las clases que implementan la lógica del negocio y de acceso a datos que se encuentran en los modelos. A continuación parte del código donde está presente:

```
class Peticion(models.Model):
    name = models.CharField(max_length=30)
    email = models.EmailField()
    company = models.CharField(max_length=30)
    max_devices = models.IntegerField()
    duration = models.IntegerField()
```

Ilustración 4 Patrón Bajo Acoplamiento.

Fuente: Elaboración Propia.

Alta Cohesión: cada elemento del diseño debe realizar una labor única dentro del sistema, no desempeñada por el resto de los elementos y auto-identificable. Ejemplos de una baja cohesión son clases que hacen demasiadas funciones. En todas las metodologías se considera la refactorización. Uno de los elementos a refactorizar son las clases saturadas de métodos. Ejemplos de buen diseño se producen cuando se crean los denominados “paquetes de servicio” o clases agrupadas por funcionalidades que son fácilmente reutilizables (bien por uso directo o por herencia) (49).

Una de las características principales del marco de trabajo Django es la organización del trabajo en cuanto a la estructura del proyecto. Permite que el software sea flexible a cambios sustanciales con efecto mínimo. Este patrón se utiliza debido a que a cada una de las clases se le asignaron responsabilidades de tal forma que estén estrechamente relacionadas entre sí y no realicen un trabajo excesivo (49). Se emplea en las clases programadas dentro del archivo Views de Django ya que fueron asignadas responsabilidades a las clases de forma tal que la cohesión siguiera siendo

alta, o sea cada clase se encargara de realizar solamente las funciones que estén en correspondencia con la responsabilidad que posea.

Experto: Este patrón se basa en la asignación de responsabilidades, las cuales se asignan a las clases que posean la información necesaria para llevarlas a cabo. Es utilizado en la capa de abstracción del modelo de datos. Con el uso del ObjectRelationalMapping (ORM) de Django, se crean las clases que representan las entidades del modelo de datos. Asociado a cada una de estas clases, son generadas un conjunto de funcionalidades que las relacionan de forma directa con la entidad que representan. Las clases contienen toda la información necesaria de la tabla que representan en la base de datos.

Este patrón es aplicado en todas las clases debido a que cada una de ellas es experta en brindar información que solo ella contiene; facilitando así el entendimiento, extensión y mantenimiento del sistema.

Controlador: Es un patrón que sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa, de tal forma que es el controlador quien recibe los datos del usuario y quien los envía a las distintas clases según el método llamado(50).
Este patrón se evidencia en la clase LicenseSerializer, ver figura 4 a continuación

```
class LicenseSerializer(serializers.HyperlinkedModelSerializer):
    time_remain = serializers.SerializerMethodField()
    peticion = PeticionSerializer(read_only=True)

    def get_time_remain(self, obj):
        return str((obj.end-now()).days)+' dias'

    class Meta:
        model = License
        fields = ['uuid', 'peticion', 'start', 'end', 'time_remain', 'state']
```

Ilustración 5 Patrón Controlador.

Fuente: Elaboración Propia.

Este patrón sugiere que la lógica de negocio debe estar separada de la capa de presentación, lo que aumenta la reutilización de código y permite a la vez tener un mayor control. Hay muchas arquitecturas de aplicación que se basan en esto, (MVC, MVVM y MVVMP). Se recomienda dividir los eventos

del sistema en el mayor número de controladores para poder aumentar así la cohesión y disminuir el acoplamiento(50).

2.5.2 Patrones GOF (Gang of Four)

Describen 23 patrones de diseño comúnmente utilizados y de gran aplicabilidad en problemas de diseño usando modelado UML. Se clasifican en tres categorías basadas en su propósito: creacionales, estructurales y de comportamiento (50).

Decorador: Es un patrón estructural que extiende la funcionalidad de un objeto dinámicamente, de tal modo que es transparente a sus clientes, utilizando una instancia de una subclase de la clase original que delega las operaciones al objeto original. Provee una alternativa muy flexible para agregar funcionalidad a una clase (50).

Este patrón se evidencia en el siguiente fragmento de código:

```
19  @api_view(['GET'])
20  def license_details(request, pk):
21      licence = get_object_or_404(License, uuid=pk)
22
23      serializer = LicenseSerializer(licence)
24      return Response(serializer.data)
```

Ilustración 6 Patrón Decorador.

Fuente: Elaboración propia.

Constructor: Es un constructor virtual que abstrae el proceso de creación de un objeto completo, centralizando dicho proceso en un único punto(50).

Este patrón se evidencia en el siguiente fragmento de código:

```python
class License(models.Model):
    uuid = models.UUIDField(
        primary_key=True, editable=False, default=uuid.uuid4,)
    peticion = models.OneToOneField(
        Peticion, on_delete=models.CASCADE, unique=True)
    start = models.DateTimeField(default=now)
    end = models.DateTimeField(blank=True, null=True, editable=False)
    state = models.CharField(max_length=10, choices=[(
        'ACTIVE', 'Active'), ('PENDING', 'Pending'), ('ENDED', 'Ended')])

    def save(self, *args, **kwargs):
        if not self.end:
            self.end = timedelta(weeks=52.1429*self.peticion.duration)+now()
        super(License, self).save(*args, **kwargs)
```

Ilustración 7 Patrón Constructor.

Fuente: Elaboración propia.

2.6 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 (51).

Django crea la asociación (o el mapeo) de las tablas de la base de datos a estructuras y tipos de datos de Python usando el modelo de datos. Un modelo es una clase que corresponde al nombre de una tabla en la base de datos, y cada variable de clase es un campo de dicha tabla. En esta clase también se especifican los índices de la tabla, llaves foráneas, ordenamiento de los datos y reglas de validación. Cada ap. del sistema Django tiene un modelo asociado, y las tablas creadas son del tipo app_nombremodelo; así que, si se cuenta con una appaccounts y el modelo tiene la clase Userporfile, la tabla se llamará accounts_userporfile. Cuando el modelo está definido, Django usa las migraciones para crear las tablas y campos necesarios en la base de datos, y si posteriormente se hacen las modificaciones al modelo, las migraciones se encargan de replicar estos cambios (33). A continuación, se muestra el modelo de datos correspondiente a la solución propuesta:

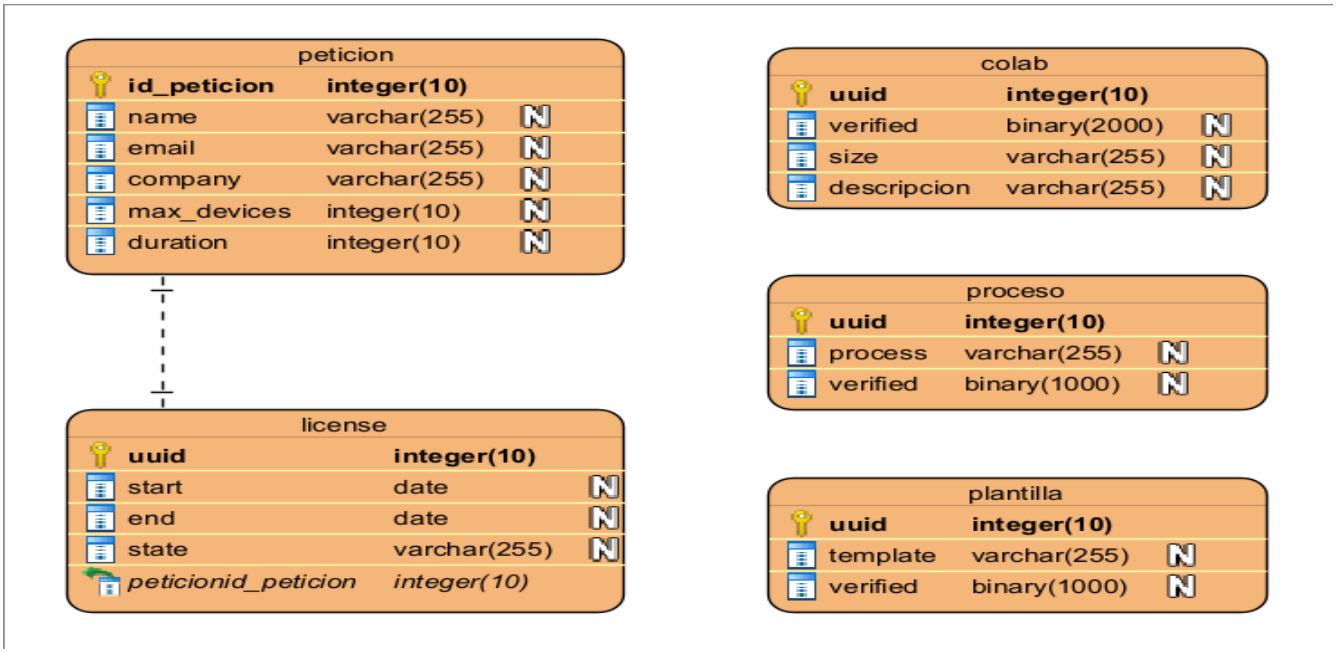

Ilustración 8 Diagrama para el Modelo de Datos.

Fuente: Elaboración propia.

2.7 Conclusiones del capítulo

Una vez definidas las directivas del desarrollo del componente para la administración de licenciamiento y colaboración de AUDAT el modelo del dominio nos permitió conocer y comprender los principales conceptos asociados al producto. Se realizó el levantamiento de los requisitos, funcionales y no funcionales, con los cuales debía de cumplir el sistema. La descripción de los requisitos funcionales fue apoyada en las historias de usuario, definiendo las funcionalidades del sistema, lo que permitió al programador una mayor claridad a la hora de codificar. El diseño del diagrama de clases, la definición de la arquitectura de desarrollo, la representación de los principales patrones de diseño (GRASP y GOF) permitieron obtener una idea concisa del componente para la administración de licenciamiento y colaboración de AUDAT, los objetivos que persigue y los elementos fundamentales a tener en cuenta en el desarrollo de la solución.

CAPÍTULO 3: IMPLEMENTACIÓN Y VALIDACIÓN DE LA SOLUCIÓN PROPUESTA.

Una vez detallada la propuesta de solución se procedió a su ejecución. El presente capítulo precisa cada uno de los pasos que desde ese momento tuvo lugar en el afán de continuar y reforzar el aseguramiento de la calidad, iniciado desde la misma planeación del proceso de software. Como parte de la vista de implementación del sistema, se obtienen el diagrama de componente de la historia de usuario que se describirá. Se ofrece un análisis de cada uno de los niveles de pruebas ejecutados, los términos en que se realizaron las pruebas y los resultados alcanzados.

3.1 Diagrama de componentes

Una vez terminado el diseño y análisis de la propuesta de solución se prosigue a diseñar la vista de implementación de la propuesta de solución a través del diseño a nivel de componentes. En esta etapa se ha establecido la estructura general de los datos y del programa del software. El objetivo es traducir el modelo del diseño a software operativo. Pero el nivel de abstracción del modelo de diseño existente es relativamente alto y el del programa operativo es bajo (30).

Los componentes son partes modulares de un sistema independientes entre sí, que pueden reemplazarse con componentes equivalentes. Son autocontenidos y encapsulan estructuras de cualquier grado de complejidad. Los elementos encapsulados solo se comunican con los otros a través de interfaces. Los componentes no solo pueden proporcionar sus propias interfaces, sino que también pueden utilizar las interfaces de otros componentes, por ejemplo, para acceder a sus funciones y servicios. A su vez, las interfaces de un diagrama de componentes documentan las relaciones y dependencias en una arquitectura de software (52). Para un mayor entendimiento, a continuación, se describe el diagrama de componentes relacionado con la implementación de la HU_5 Exportar documentos

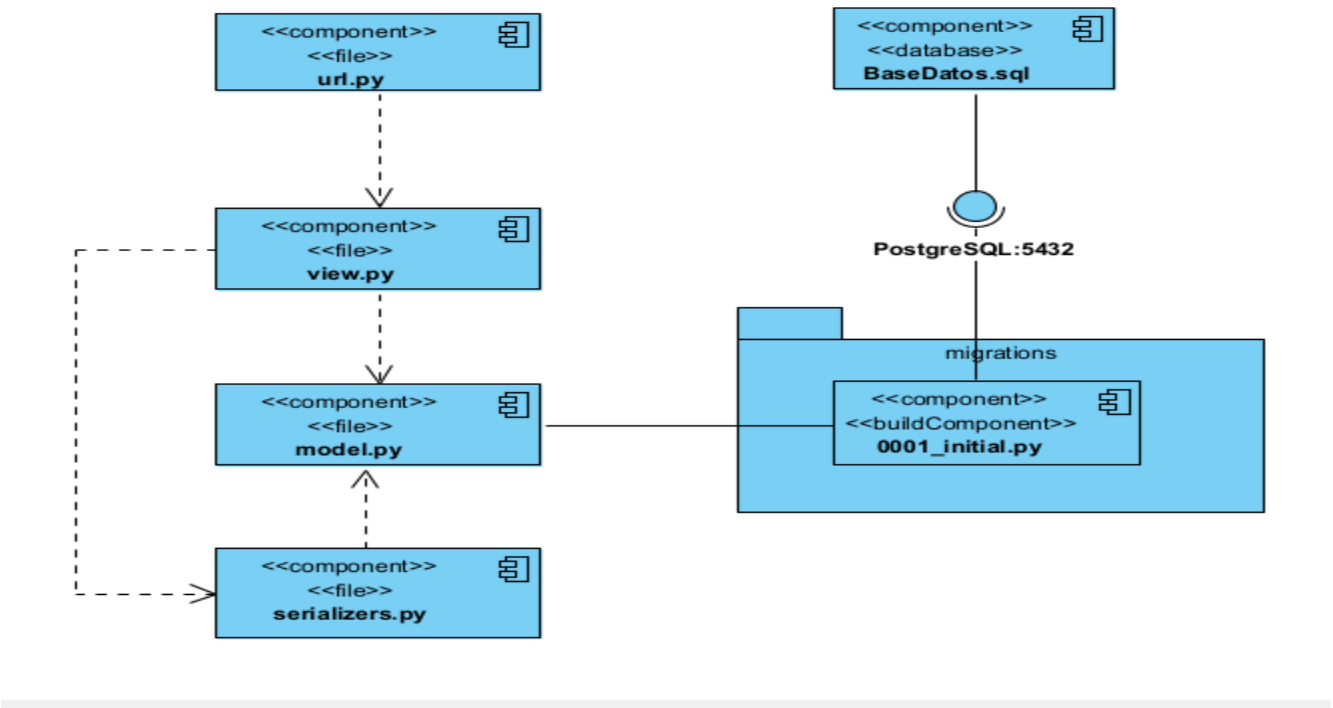

Ilustración 9 Diagrama de Componentes.

Fuente: Elaboración propia.

3.2 Estándar de codificación

Un estándar de codificación completo comprende todos los aspectos de la generación de código. Si bien los programadores deben implementar un estándar de forma prudente, este debe tender siempre a lo práctico. Un código fuente completo debe reflejar un estilo armonioso, como si un único programador hubiera escrito todo el código de una sola vez. Una faceta del desarrollo de software en la que todos los programadores influyen especialmente es en la técnica de codificación. “El mejor método para asegurarse de que un equipo de programadores mantenga un código de calidad es establecer un estándar de codificación”(53). Para el desarrollo del componente para la administración de licenciamiento y colaboración de AUDAT se utilizó el estándar de codificación PEP 8 de Python. (54).

```
@api_view(['GET'])
def get_valid_licence(request, pk):
    queryset = License.objects.get(uuid=pk)
    txt = '\n'.join([queryset.peticion.name, queryset.peticion.email, queryset.peticion.company, str(queryset.start.strftime(
                    "%Y%m%d%H%M%S")), str(queryset.end.strftime("%Y%m%d%H%M%S")), str(queryset.peticion.duration), queryset.state])
    coder=AES_pkcs5()
    lsc=coder.encrypt(txt)
    response = HttpResponse(lsc)
    response['Content-Disposition'] = 'attachment; filename="licenses.afe"'
    return response
```

Ilustración 10 Estándar de codificación.

Fuente: Elaboración propia.

En el fragmento de código anterior se pueden evidenciar algunas de las reglas definidas en el estándar PEP 8 como:

- Uso de 4 espacios de identación por línea de código.
- Separar las definiciones de funciones con dos líneas en blanco.
- No pasar de los 79 caracteres por línea de código.

3.3 Validación de Requisitos

Con el objetivo de ratificar que los requisitos del software obtenidos definen el sistema que el cliente desea, se llevó a cabo un proceso de validación de estos, para el cual se emplearon las siguientes técnicas:

3.3.1 Revisión técnica formal

El primer mecanismo para la validación de los requisitos es la revisión técnica formal (RTF). El equipo de revisión incluye ingenieros del sistema, clientes, usuarios, y otros intervinientes que examinan la especificación del sistema buscando errores en el contenido o en la interpretación, áreas donde se necesitan aclaraciones, información incompleta, inconsistencias (es un problema importante), requisitos contradictorios, o requisitos imposibles o inalcanzables (30).

Aunque la validación de requisitos puede guiarse de manera que se descubran errores, es útil chequear cada requisito con un cuestionario. Las siguientes cuestiones representan un pequeño subconjunto de las preguntas que pueden plantearse:

- ¿Está el requisito claramente definido?

- ¿Pueden interpretarse mal?
- ¿Está identificado el origen del requisito (por ejemplo: persona, norma, documento)?
- ¿El planteamiento final del requisito ha sido contrastado con la fuente original?
- ¿El requisito está delimitado en términos cuantitativos?

Las preguntas planteadas en la lista de comprobación ayudan asegurar que el equipo de validación dispone de lo necesario para realizar una revisión completa de cada requisito (30). Para llevar a cabo este proceso de validación se aplicó la métrica Calidad de la especificación, que se describe a continuación.

Métrica Calidad de la especificación: el empleo de esta métrica permite obtener un alto nivel de entendimiento y precisión de los requisitos según (54), para llegar a ello, se debe primeramente calcular el total de requisitos de software como se muestra a continuación:

$\boldsymbol{Nr}$: el total de requisitos de especificación.

$\boldsymbol{Nf}$: cantidad de requisitos funcionales.

$\boldsymbol{Nnf}$: cantidad de requisitos no funcionales.

$$\boldsymbol{Nr} = \boldsymbol{Nf} + \boldsymbol{Nnf}$$

Como resultado de la sustitución de los valores, para la presente investigación se obtiene:

$$\boldsymbol{Nr} = \mathbf{6} + \mathbf{6}$$

$$\boldsymbol{Nr} = \mathbf{12}$$

Para determinar, finalmente, la Especificidad de los Requisitos (ER) o ausencia de ambigüedad en los mismos se realiza la siguiente operación:

$$\boldsymbol{ER} = \boldsymbol{Nui} \,/\, \boldsymbol{Nr}$$

Donde $\boldsymbol{Nui}$ es el número de requisitos para los cuales todos los revisores tuvieron interpretaciones idénticas. Mientras más cerca de 1 esté el valor de $\boldsymbol{ER}$, menor será la ambigüedad.

Para el caso de los requisitos obtenidos en la solución, 5 produjeron contradicción en las interpretaciones. Sustituyendo las variables se obtiene:

$$\boldsymbol{ER} = \mathbf{9} \ / \ \mathbf{12}$$

$$\boldsymbol{ER} = \mathbf{0.75}$$

Indicando que el grado de ambigüedad de los requisitos es (25%) ya que el 75% son entendibles. Los requisitos ambiguos fueron modificados y validados para garantizar una correcta interpretación.

3.4 Estrategia de pruebas

Una estrategia de prueba de software proporciona una guía que describe los pasos que deben realizarse como parte de la prueba. Debe ser lo suficientemente flexible para promover un uso personalizado de la prueba(56). Al probar el software se verifican los resultados de la prueba que se opera para buscar errores, anomalías o información de atributos no funcionales del programa(57).

El proceso de prueba tiene dos metas distintas:

- Demostrar al desarrollador y al cliente que el software cumple con los requerimientos.
- Encontrar situaciones donde el comportamiento del software sea incorrecto, indeseable o no esté de acuerdo con su especificación.

Una estrategia de prueba está compuesta por niveles, tipos, métodos y técnicas de pruebas, así como los casos de prueba (58).

3.4.1 Pruebas Unitarias

Este nivel de prueba está enfoca a los esfuerzos de verificación en la unidad más pequeña del diseño de software: el componente. Se orientan al procesamiento interno del software y las estructuras de datos dentro de las fronteras de un componente. Para la conformación de las pruebas unitarias se emplea el método de caja blanca, las cuales están dirigidas a evaluar las funciones internas del software; basándose en el examen cercano de los detalles de procedimiento del software. Las rutas lógicas a través del software y las colaboraciones entre componentes se ponen a prueba al revisar conjuntos específicos de condiciones y/o bucles (30).

Para realizar las pruebas en este nivel, se utilizó el módulo unittest integrado en la biblioteca estándar de Python. Debido a los módulos y herramientas que posee, se puede ejecutar un test para comprobar las funciones del código, encontrar errores y facilitar el desarrollo del componente (59). En la siguiente imagen se muestra los resultados obtenidos en una de las pruebas realizadas el cual arrojó resultados positivos.

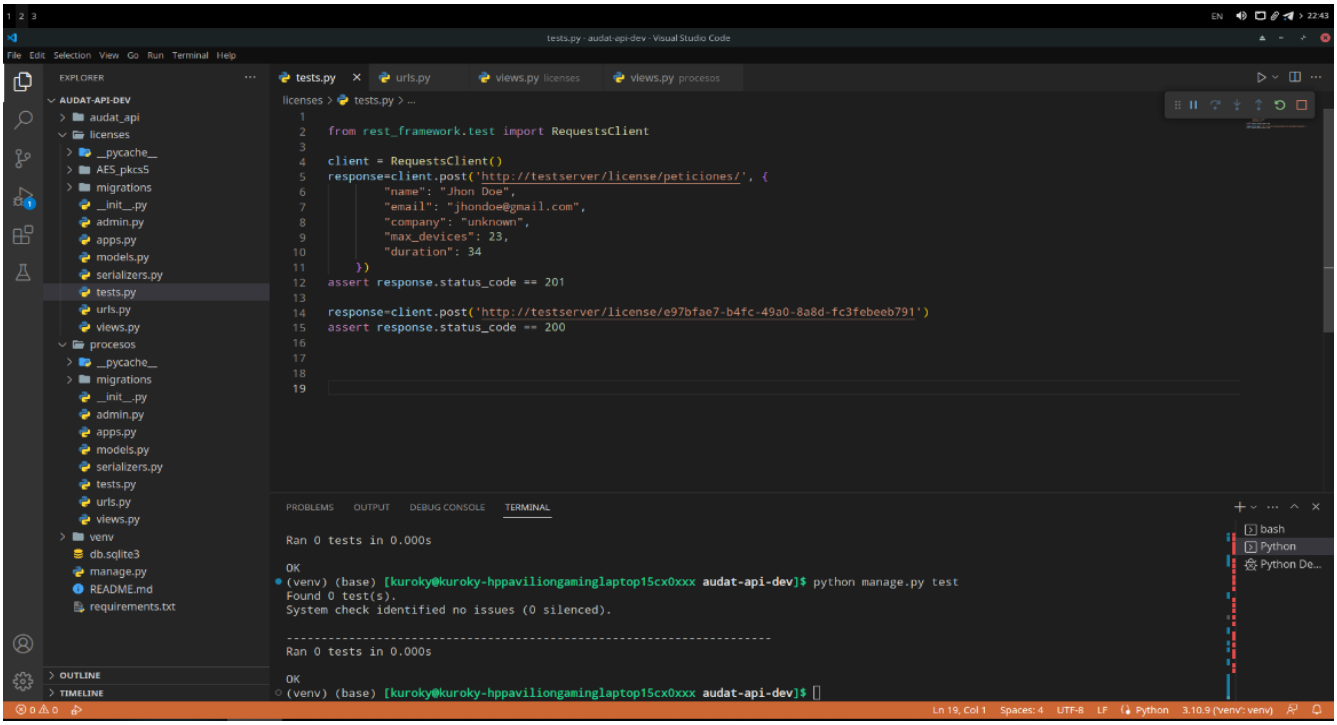

Ilustración 11 Pruebas a nivel de unidad, resultado de la unnitest.

Fuente: Elaboración propia.

3.4.2 Pruebas de sistema

Esta prueba tiene como objetivo verificar que se han integrado adecuadamente todos los elementos del sistema y que realizan las operaciones apropiadas funcionando como un todo (60).

3.4.2.1 Prueba de seguridad

La prueba de seguridad está enfocada al ámbito de la aplicación, asegurando que los usuarios solo accedan cuando estén registrados en el sistema.

Resultados: Para la realización de las pruebas de seguridad al componente se creó un usuario y se procedió con las pruebas. Fueron intentos de violación de la seguridad del sistema, tratando de acceder a este con usuarios no existentes o utilizando usuarios reales con contraseñas erróneas. Estas pruebas de acceso arrojaron resultados favorables al sistema, ya que no se logró acceder con usuarios no registrados previamente ni con falsas contraseñas, demostrando que el componente cuenta con un mecanismo de autenticación seguro, garantizando que solamente trabajen con el programa personas o equipos autorizados previamente.

3.4.2.2 Rendimiento

Se aplicaron pruebas de carga y rendimiento con la herramienta de prueba Apache JMeter se realizaron las pruebas con 50 usuarios concurrentes. El sistema tuvo un tiempo de respuesta entre 600ms y 2s sin presentar caídas con un valor medio de 970ms y una desviación de 465ms.

Las prestaciones de hardware que posee la PC cliente y el servidor del software donde se realizaron las pruebas se muestran a continuación:

PC cliente:

- Ordenador Intel Core 2 Duo, con 2.0 GHz de velocidad de microprocesador.
- Memoria RAM de 4 GB.

PC servidor:

- Ordenador Intel Core i7, con 3.4 GHz de velocidad de microprocesador.
- Memoria RAM de 8 GB.

Muestra #	Tiempo de comie...	Nombre del hilo	Etiqueta	Tiempo de Muest...	Estado
1	07:04:53.149	Usuarios 1-1	getDocList	177	✓
2	07:04:53.249	Usuarios 1-2	getDocList	252	✓
3	07:04:53.349	Usuarios 1-3	getDocList	553	✓
4	07:04:53.449	Usuarios 1-4	getDocList	756	✓
5	07:04:53.550	Usuarios 1-5	getDocList	1163	✓
6	07:04:53.650	Usuarios 1-6	getDocList	1267	✓

Ilustración 12 Captura a Apache JMeter luego de correr las pruebas de rendimiento.

Fuente: Elaboración propia.

3.4.3 Prueba de aceptación

La prueba de aceptación de sistema por parte del personal de operaciones o de la administración del sistema se realiza, por lo general, en un entorno de producción (simulado) (60).

Las pruebas de aceptación se realizaron a especialistas del centro CREAD vinculados con el producto de software AUDAT, dentro de ellos: Reinaldo Álvarez Luna como Jefe del centro CREAD, Liuba María Infante Infante y Miguel Ángel Nápoles Molina, a través de las entrevistas semi-estructuradas. Se probaron todas las funcionalidades del componente y se verificó que todas cumplían con los requisitos definidos, esto manifestó un 97.2% de conformidad. Se deja constancia de la emisión del acta de aceptación oficial firmada y acuñada por el Jefe del Centro (ver anexo # 2).

3.5 Conclusiones del capítulo

Al finalizar el presente capítulo se arribó a las siguientes conclusiones parciales:

- Definir una estrategia de pruebas contribuyó a un temprano descubrimiento de errores en el sistema.
- Luego de cada cambio significativo en el sistema, se realizaron pruebas de seguridad a las principales funcionalidades, exigiéndose un 100 % de efectividad para cada una.

- Las pruebas de aceptación permitieron validar la solución, y verificar que se cumplieron los requisitos identificados por el cliente y el equipo de desarrollo.

CONCLUSIONES

Finalizada la presente investigación se puede concluir que se desarrollaron todas las tareas a fin de cumplir los objetivos propuestos, resaltando que:

- El estudio de la bibliografía y análisis de los sistemas existentes de Programas de Licenciamiento demostró la necesidad del desarrollo de un componente para la administración de licenciamiento y colaboración del producto de software AUDAT.
- Durante la planificación y el diseño de la solución se obtuvieron los artefactos propuestos por la metodología de desarrollo seleccionada, lo que permitió facilitar la implementación de las funcionalidades definidas.
- La implementación de la propuesta de solución contribuye a la gestión del proceso de administración de licenciamiento y colaboración del producto de software AUDAT.
- Las pruebas de software realizadas permitieron garantizar un correcto funcionamiento del componente, así como el cumplimiento de las necesidades y requisitos del cliente.

Referencias bibliográficas

REFERENCIAS BIBLIOGRÁFICAS

1. VALDAVIDA, MAGDALENA CORDERO. Auditoría digital: el reto del siglo XXI . 2018.

2. ZAMBRANO, LUIS ORLANDO ALBARRACIN. AUDITORIA INFORMATICA. 2021.

3. MARRERO, MANUEL MARRERO. LA AUDITORÍA DE GESTIÓN, UNA ALTERNATIVA PARA LOS AUDITORES INTERNOS EN LAS EMPRESAS CUBANAS. 2018.

4. www.parlamentocubano.gob.cu. proyecto de ley de la CGR. [En línea] 2022. https://www.parlamentocubano.gob.cu/sites/default/files/documento/2022-10/Proyecto%20de%20Ley%20de%20la%20CGR.pdf) .

5. http://www.parlamentocubano.cu. [En línea] 2009. http://www.parlamentocubano.cu/index.php/labor-legilativa/leyes%20/318-ley-no-107-de-la-contraloria-general-de-la-republica-de-cuba.html%20.

6. Proyecto Ley de la Contralorí General de la República de Cuba y del sistema de control superior de los fondos públicos y de la gestión administrativa . Popular, Asamble Nacional del Poder. La Habana : s.n., 2022.

7. GARNACHE, ALBERTO MENDOZA. PROPUESTA DE MECANISMO DE LICENCIAMIENTO PARA EL SISTEMA DE AUDITORÍA DE DATOS. 2020.

8. ENRÍQUEZ, ARLYNE MEDINA. FUNDAMENTOS TEÓRICO-CONCEPTUALES DE LA AUDITORÍA DE PROCESOS . 2021.

9. NOGUEIRA, YULY E MEDINA. FUNDAMENTOS TEÓRICO-CONCEPTUALES DE LA AUDITORÍA DE PROCESOS . 2021.

10. ALBORNOZ, M. CLAUDIA. INTERFAZ GRÁFICA DE USUARIO: EL USUARIO COMO PROTAGONISTA DEL DISEÑO. 2021.

11. BALCÁZAR-RENGIFO, ALBERTO. INTERFAZ COLABORATIVA Y EMOCIONAL PARA INTERPRETAR EL SENTIDO COMÚN. 2018.

12. PEREZ, MANUEL J. COMO PROTEGER MI SOFTWARE. 2021.

Referencias bibliográficas

13. BLANCO, YUDI CASTRO. (CARACTERIZACIÓN DE LAS LICENCIAS DE SOFTWARE, PAUTAS PARA SU ENSEÑANZA EN CARRERAS DE INFORMÁTICA . 2019.

14. MATEOS, ANGEL DEL RIO. INTRODUCCION A LA CRIPTOLOGIA. 2021.

15. VELANDIA, LUCY NOEMY MEDINA. CRIPTOGRAFÍA Y MECANISMOS DE SEGURIDAD . 2017.

16. Barrera, Jamie Areli Hasimoto. Esquema de marca de agua robusto ante ataques geometricos para la identificacion de imagenes con derechos de autor. 2020.

17. iris, red. iriscert. [En línea] 2020. rediris.es.

18. Leyanis Martinez Vence, Dayanis Maria Rodriguez Hernandez, Danna Fumero Moreno. Revista Iberoamericana de la propiedad intelectual. [En línea] 22 de junio de 2023.

19. CARREÑO, EDWIN JOAO MERCHÁN. CARACTERIZACIÓN DE LAS LICENCIAS DE SOFTWARE, PAUTAS PARA SU ENSEÑANZA EN CARRERAS DE INFORMÁTICA. 2021.

20. www.gacetaoficial.cu. [En línea] (http://www.gacetaoficial.cu/html/leyes-espanol.html.).

21. www.ocpi.cu . [En línea] (http://www.ocpi.cu/.

22. Salinas, Silvia Alicia. Software colaborativo de trabajo. 2018.

23. Citrix Service Provider. [En línea]

24. Microsoft SPLA. [En línea]

25. Symantec ExSP Program. [En línea]

26. VMware Cloud Provider Program. [En línea]

27. METODOLOGÍAS ÁGILES VS TRADICIONALES . MORY, FABIO J. 2021, REVISTA EMPRESARIAL & LABORAL.

28. YESSICA OSUNA, SANTIAGO ROUGOSKI. (METODOLOGÍAS ÁGILES PROCESO UNIFICADO ÁGIL (AUP). 2019.

29. RODRIGUEZ, TAMARA SANCHEZ. METODOLOGÍA DE DESARROLLO PARA LA ACTIVIDAD PRODUCTIVA DE LA UCI. 2015.

30. Visual paradigm for uml, tool for software application development . 2013.

31. Pressman, Roger S. 2020.

32. Domínguez, Dorado M. Todo Programación. Madrid : Editorial Iberprensa, 2018.

33. Rossum, Guido van. El tutorial de Python. . 2019.

34. DJANGO The Web framework for perfectionists with deadlines. DJANGO The Web framework for perfectionists with deadlines. . [En línea] 2022. https://www.djangoproject.com/. .

35. Powerdata. 2019. blog.powerdata.es. [En línea] 9 de mayo de 2019. [Citado el: 2023 de octubre de 11.] https://blog.powerdata.es/el-valor-de-la-gestion-de-datos/que-es-un-gestor-de-datosy-para-que-sirve..

36. Ordóñez, M. P. Z., Ríos, J. R. M., & Castillo. Administración de Bases de datos con PostgreSQL. 2017. Vol. 19.

37. Pérez Villazón, Yoandy. Estrategia para la migración a aplicaciones de código abierto. Trabajo final presentado en opción al título de Máster en Informática Aplicada. La Habana : s.n., 2015.

38. Khatri, Vijay;. Geekflare. [En línea] 27 de septiembre de 2023. geekflare.com.

39. F. Javier Díaz, Claudia M. Tzancoff Banchoff, Anahí S. Rodriguez, Valeria Soria. Usando Jmeter para pruebas de rendimiento. 2020.

40. GUTIÉRREZ, MAURICIO ERNESTO. INTRODUCCIÓN A LA INGENIERÍA DEL SOFTWARE. ESPECIFICACIÓN DE REQUISITOS. 2019.

41. MCGRAW, CRAIG LARMAN . HILL. UML Y PATRONES INTRODUCCIÓN AL ANÁLISIS Y DISEÑO ORIENTADO A OBJETOS. 2017.

42. LOS REQUISITOS NO FUNCIONALES DE SOFTWARE. UNA ESTRATEGIA PARA SU DESARROLLO EN EL CENTRO DE INFORMÁTICA MÉDICA. MOLINA HERNÁNDEZ, YENISEL, AILEC GRANDA DIHIGO, ALIONUSKA VELÁZQUEZ CINTRA. 2019, Vol. 13.

43. LA NORMA ISO/IEC 25000 Y EL PROYECTO KEMIS PARA SU AUTOMATIZACIÓN CON SOFTWARE LIBRE REICIS. . MARCOS JOSÉ ARROYO, ALICIA GARZÁS, MARIO JAVIER PIATTINI. NÚM. 2, España : s.n., Vol. VOL. 4.

44. asana.com. [En línea] https://asana.com/es/resources/user-stories#%C2%BFQu%C3%A9.

45. BLANCARTEITURRALDE, OSCAR JAVIER. INTRODUCIÓN A LOS PATRONES DE DISEÑO, UN ENFOQUE PRACTICO.

46. PRESSMAN, ROGER. INGENIERÍA DE SOFTWARE: UN ENFOQUE PRÁCTICO. NUEVA YORK, EUA : s.n., 2007.

47. ANÁLISIS COMPARATIVO DE PATRONES DE DISEÑO DE SOFTWARE . OSCAR DANILO GAVILÁNEZ ALVAREZ, NATALIA LAYEDRA VINICIO RAMOS. 2022.

48. www.django.com. [En línea] https://www.django-rest-framework.org/.

49. LINEAMIENTOS PARA EL DISEÑO DE APLICACIONES WEB SOPORTADOS EN PATRONES GRASP. ORTEGA, GILBERTO ANDRÉS VARGAS. 2021.

50. LARMAN, CRAIG. MODELO DE DOMINIO. UML Y PATRONES. . 2019.

51. datos., Tecnologías-Información. Modelo de. [En línea] 2018. https://www.tecnologias-informacion.com/modeladodatos.html..

52. IONOS. Digital Guide. [En línea] 23 de septiembre de 2020. https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/diagrama-de-componentes/.

53. DOCS, G. RECUPERADO EL ESTÁNDAR DE CODIFICACIÓN. [En línea] https://docs.google.com/document/d/1rbxDFM0zsbFDNRZeM2FoXfRDbYSiSt6tCdbYPA0qdzs/edit?#!.

54. PYTHON ENHANCEMENTS PROPOSAL PEP 8 – STYLE GUIDE FOR PYTHON CODE. . [En línea] HTTPS://PEPS.PYTHON.ORG/PEP-0008/..

55. Somerville, Ian. Software Engineering. 9na edicion. 2011.

56. Pressman, Roger S. 2010.

Referencias bibliográficas

57. SOMMERVILLE, IAN. Software Engineering. Última Edición. 2015.

58. KeepCoding, Redacción. KeepCoding Tech School. [En línea] keepcoding.io.

59. Sommerville, Ian. Software Engineering: International Edition. 2011.

60. PROBADOR CERTIFICADO DEL ISTQB®. 2018.

61. Verificación y validación de requerimientos por equipos de producto. visuresolution.com. 2022.

ANEXOS

Anexo # 1 Carta de Aceptación para el componente para la administración del licenciamiento y colaboración de AUDAT (firmada y validada por el Jefe del Centro CREAD: Reinaldo Álvarez Luna).

UCI Universidad de las Ciencias Informáticas

CARTA DE ACEPTACIÓN

En cumplimiento con la fase de desarrollo y en función de la ejecución del trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas: Componente para la administración de licenciamiento y colaboración de AUDAT, se hace entrega de los productos que se relacionan a continuación:

-Componente de licenciamiento y colaboración (código fuente)

Entregado por:

X _____

María del Carmen Hernández Landestoy
Tesista

Recibido por:

X _____

Reinaldo Álvarez Luna
Jefe del centro CREAD

DATEC CENTRO DE TECNOLOGÍAS Y GESTIÓN DE DATOS

Fecha: 2/11/2023

Anexos

Anexo # 2 Entrevista semi-estructurada aplicada a los especialistas del centro CREAD para probar todas las funcionalidades del componente y se verificar que todas cumplían con los requisitos definidos.

Esta entrevista es considerada personal, dirigida a los especialistas del centro CREAD vinculados al producto de software AUDAT.

El objetivo de la encuesta es determinar el cumplimiento de todos los Requisitos Funcionales, así como el tiempo que tarda en responder el sistema. Es muy importante para el perfeccionamiento de nuestro trabajo.

Por favor complete la encuesta cuidadosamente al leerla primero, y luego señale sus respuestas con una “X”.

1. ¿Considera que estén implementados todos los requisitos expuestos en el trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas “Componente para la administración de licenciamiento y colaboración de AUDAT”?.

   a) __ SI b) __ NO

   1.1 En el caso de que su respuesta sea negativa exponga cuál o cuáles requisitos considera que no estén implementados.

   _____

   _____

   _____

2. ¿En qué porciento (%) considera que estén bien implementados todos los requisitos? Marque con una X en la opción que más se aproxime a su respuesta.

   a) _ Menos de 50 % de efectividad.

   b) _ De 50% a 70 % de efectividad.

   c) _ De 70 % a 90% de efectividad.

   d) _ 100 % de efectividad.

3. ¿Qué tiempo demoró el sistema en responder a las peticiones que le realizó?

a) _ menos de 1 segundo.

b) _ de 1 a 3 segundos.

c) _ más de 3 segundos.