UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICA

FACULTAD 1

API REST asociada al repositorio de Nova

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

Autora: Dareyna de la Caridad Muñoz González

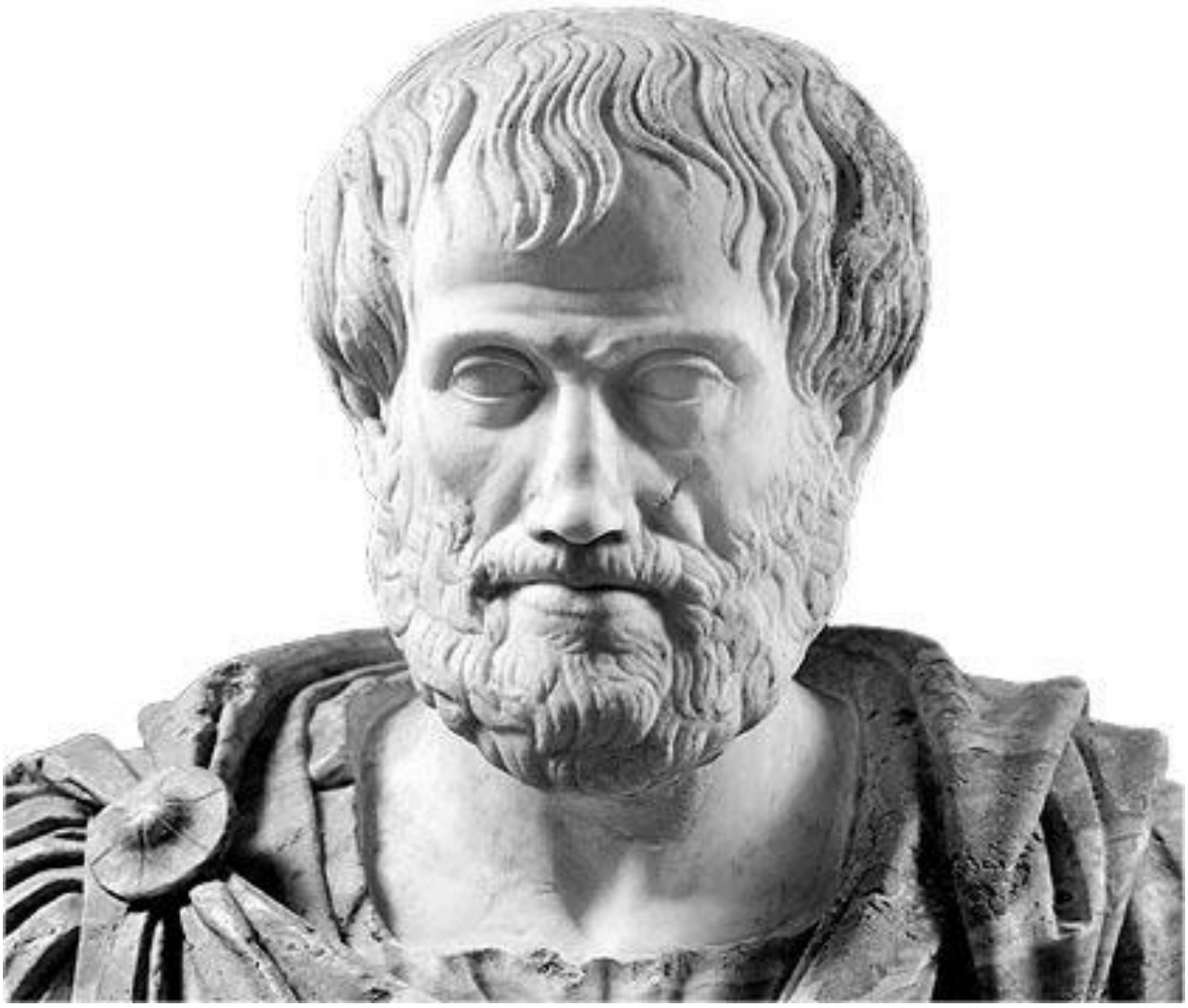
Tutores:

M.Sc. Yoandy Pérez Villazón

Ing. Neyvis González Trejo

LA HABANA, JUNIO DEL 2018

"AÑO 60 DE LA REVOLUCIÓN"



"La inteligencia consiste no sólo en el conocimiento, sino también en la destreza de aplicar los conocimientos en la práctica".

Aristóteles (384 AC-322 AC) Filósofo griego

AGRADECIMIENTOS

A mi abuela Caridad que donde quiera que te encuentres te dedico de todo corazón y con amor este título por ser una de las personas más importantes en mi vida. Te amo y no sabes cuánto diera porque estuvieras a mi lado, solo me queda decirte gracias por haberme dado amor y cariño que dios te tenga en la gloria te amo.

A mi abuelo Vicente por habernos inculcado amor, el valor del trabajo, el sacrificio que debemos hacer para alcanzar un propósito en la vida, hoy te digo que eres un ejemplo en mi vida y donde quiera que te encuentres, siempre vas a caminar a mi lado te amo abuelo y que dios te tenga en la gloria.

A mi abuela Hilda decirte que no tuvimos tanto tiempo para compartir, pero eso no significa nada con el cariño y el amor que siento por ti, sé que donde quiera que estés vas a estar orgullosa de mí, te quiero.

A mi madrina Cándida Cordoví qué te pudiera decir, que desearía desde el fondo de mi alma que estuvieras conmigo compartiendo este lindo sueño que, con tanto sacrificio y amor, he luchado por alcanzarlo. Tú que ya me querías sin haber llegado a este mundo y sin decir mis primeras palabras, y me acogiste en tus brazos y no me soltaste nunca, aunque han pasado los años y no te encuentres ya en este mundo te siento tan presente y te digo desde el fondo de mi alma, gracias por haberme dejado entrar en tu vida, te quiero.

A mi madre Nereyda a mi negra hermosa, gracias por ser esa madre especial y dedicada te debo todo lo que soy y lo que llegaré a ser. Mami en este día tan especial no es solo para mí, sino hoy cumplimos un sueño juntas, gracias por no soltarme la mano desde que di mis primeros pasos y por tenerla sujetada hasta el día de hoy, por el sacrificio, por las angustias que tantas veces pasamos, te debo esto y mucho más y no me alcanzan las líneas para decirte lo importante que eres para mí y si no lo sabes pues te lo digo, no me cansaré de decirte que te amo y que eres lo más importante en mi vida te quiero mi prieta.

A mi padre, mi feo qué te puedo decir mil gracias por todo lo que me has dado y sé el sacrificio que has hecho para hoy estar aquí a mi lado, si de algo no te puede quedar duda es que te quiero papá y mucho. La vida me ha premiado y de la mejor forma, una de ellas es tener un padre como tú y tener la dicha de que te encuentres aquí conmigo, te amo.

A mis tíos y tías Digna, Blanca Rosa, Jose, Celia, Francisco, Emilian por todo lo que han hecho por mí, por su cariño, paciencia por no abandonarme nunca y estar en todos los momentos tanto felices como tristes, gracias por ser tan especiales y afortunada me puedo considerar por haberlos conocido y dejarme entrar en sus vidas y por ser como una hija para ustedes, los amo.

A mis hermanos Michel, Yusniel gracias por su preocupación los quiero mucho.

Agradecimientos

A Magali y Justina que donde quiera que se encuentren allá en el cielo decirles que las quise mucho beso.

A mi prima Yusleidis decirte que te quiero como una hermana, que seguí tus pasos, gracias por estar en los momentos más importantes en mi vida tú sabes que te quiero mucho, besos mi vida.

A mi primo Juan Carlos qué te puedo decir, mil gracias por estar pendiente de mí y de mis padres, por todo lo que me has ayudado, por estar en los momentos más importantes en mi vida, eres una de las personas que te voy a estar eternamente agradecida por cada gesto, por cada detalle en que has participado sin esperar nada a cambio, gracias, beso para ti y para Denia que eres la prima más risueña que he tenido besito.

A la personita que desde que nació se robó parte de mi corazón y que la quiero como mi hija, aunque hoy no entiendas mucho decirte que te amo, mi Sheyla.

A mis padrinos por acogerme como su hija por preocuparse por mí, por ser mis segundos padres les estoy agradecida por el amor que me han brindado incondicional, a ti María te digo gracias y por ese cariño sincero y el amor de madre te quiero mucho, a ti Julio mi bombón te quiero y tú lo sabes eres para mí como mi papá te adoro mi negro, a Dulce por su cariño y preocupación, a Cecilia por su cariño se te quiere mucho, a Rafael por ser mi guía, mi amigo, mi padre te quiero mucho y hoy tengo el orgullo de decir que eres como un padre para mí y siempre lo serás gracias por tu cariño.

A mis tutores Yoandy y Neyvi por su preocupación y por ser unos guías en mi vida y por permitir cumplir uno de mis sueños, gracias por todo.

A Ivaniet por ser esa persona tan especial no tengo palabras por todo lo que has hecho por mi mil gracias te quiero y lo sabes, este logro te lo debo a ti besos.

A mi oponente Nurisel por toda su dedicación beso para ti y por tu preocupación.

A Yosel y Yasiel por toda la ayuda incondicional que me han dado besos y mil gracias.

A mi grupo 1503 por tantos años de sacrificio, por acogerme y ser parte en mi vida le doy gracias y que algún día la vida nos vuelva a reunir de nuevo se les quiere.

A Leonardo mi leo lástima que no estés aquí, pero decirte que eres una persona muy especial para mí te quiero mucho y tú lo sabes, no hay mucho por hablar porque lo demás tú lo sabes te quiero.

A mi prieta Aidee por estar en las buenas y en las malas, como quien dice ya vencimos, y espero que nuestra amistad no solo llegue aquí, se te quiere.

DEDICATORIA

La presente investigación con amor y cariño se la dedico, con toda devoción a mis padres, abuelos y a mi sobrina que son la razón de vida.

DECLARACIÓN DE AUTORÍA

Declaro por este medio que yo Dareyna de la Caridad Muñoz González, con carné de identidad 94031804972 soy la autora principal del trabajo de diploma titulado API REST asociada al repositorio de Nova y autorizo a la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio, así como los derechos patrimoniales con carácter exclusivo.

Para que así conste firman la presente a los __ días del mes de junio del año 2018.

_____

Dareyna de la Caridad Muñoz González

Autora

_____ _____

M.Sc. Yoandy Pérez Villazón

Tutor

Ing. Neyvis González Trejo

Tutora

DATOS DE CONTACTO

M.Sc. Yoandy Pérez Villazón

Universidad de las Ciencias Informáticas, La Habana, Cuba

Correo: yvillazon@uci.cu

Ing. Neyvis González Trejo

Universidad de las Ciencias Informáticas, La Habana, Cuba

Correo: ngonzalez@uci.cu

RESUMEN

La presente investigación tuvo como objetivo desarrollar una API REST que ofrezca información asociada al uso del repositorio de la distribución cubana GNU/Linux Nova. Se empleó el método teórico Analítico–Sintético que permitió descomponer el problema en sus partes para poder realizar un mejor análisis, tomando información de distintas fuentes bibliográficas y la extracción de los elementos más importantes que se relacionan con los servicios web que permitan obtener información sobre el uso del repositorio de la distribución cubana de GNU/Linux Nova. Además, se utilizó el método empírico Observación que posibilitó conocer la estructura del repositorio de Nova, la agrupación de los paquetes, el ciclo de vida de un paquete, cómo se realizan las descargas de paquetes y la gestión de los *logs* del repositorio. Para guiar el proceso de construcción de la propuesta de solución se utilizó la metodología de desarrollo de software Variación de AUP para la UCI y en la implementación se emplearon tecnologías y herramientas de software libre. La evaluación de la propuesta de solución se realizó a partir de la aplicación de técnicas, encuestas y pruebas que garantizan el correcto funcionamiento de la aplicación y demostraron la satisfacción del cliente hacia la API REST desarrollada. Al finalizar la presente investigación se obtuvo una API REST que permite mostrar la información de los paquetes que contiene el repositorio, además contribuye a prestar un mejor servicio en cuanto a la información asociada a los repositorios.

Palabras claves: API REST, GNU/Linux, Nova, repositorio.

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

El desarrollo de la informática ha pasado por varias etapas, en el transcurso del mismo se ha reflejado la evolución hacia el software libre[1]. Aunque ya hace varias décadas que el software libre existe, hasta los últimos tiempos no se ha perfilado como una alternativa válida para muchos usuarios, empresas, instituciones y gobiernos. Actualmente, las distribuciones GNU/Linux cuentan con una amplia variedad de usuarios y de ámbitos de trabajo donde son utilizados. Aunque su naturaleza de software libre creó inicialmente cierta resistencia por parte de los usuarios y empresas, GNU/Linux ha demostrado estar a la altura de cualquier otro sistema operativo existente (1).

En la actualidad la mayoría de las distribuciones de GNU/Linux como *Debian, Fedora, Ubuntu*, entre otras, poseen repositorios de código fuente y binarios, donde se aloja información que representa tanto el código fuente de las aplicaciones, como los binarios que se utilizan para ser instalados en el sistema operativo del usuario. Las principales características de estos repositorios están dadas en que presentan una estructura bien definida y organizada, el acceso remoto a dicha estructura se realiza normalmente por los protocolos HTTP (*HyperText Transfer Protocol*, Protocolo de transferencia de hipertexto) y FTP (*File Transfer Protocol*, Protocolo de Transferencia de Ficheros) para la actualización e instalación desde los gestores de paquetes (2).

Hoy en día diferentes organizaciones han tomado la decisión de migrar a estándares de software libre y código abierto[2], teniendo en cuenta que con la aparición del código abierto, gran parte del mundo se ha proyectado hacia la soberanía tecnológica. Desde simples aplicaciones hasta sistemas operativos como GNU/Linux, han tomado posición en organizaciones e instituciones de todo el mundo. Uno de los productos más interesantes en cuanto a este mercado son las distribuciones de GNU/Linux compuestas por un compilado de software, de tal forma que sea fácil de descargar, instalar y usar. Esto permite lograr un sistema perfectamente usable y con muchas funcionalidades (3).

En Cuba una de las instituciones que apoyan el proceso de migración a software libre es la Universidad de las Ciencias Informáticas (UCI), mediante la transferencia de productos informáticos, a diferentes Organismos de la Administración Central del Estado, que contribuyen a la soberanía tecnológica de la sociedad cubana. La UCI cuenta con varios centros de investigación y desarrollo, entre ellos el Centro de Software Libre (CESOL), que tiene como objetivo principal la elaboración de soluciones informáticas

[1] Software libre: Programa informático que cumple con las 4 libertades expresadas en copiar, mejorar, modificar y redistribuir un software.
[2] Código abierto: Utiliza el software libre teniendo un punto de vista más orientado a los beneficios prácticos de poder acceder al código fuente, que a las cuestiones éticas y morales.

para apoyar este proceso de migración. En el centro se desarrolla la distribución cubana GNU/Linux Nova como variante de sistema operativo a utilizar en las diferentes instituciones del país.

Actualmente Nova cuenta con un centro de software que dispone de varias secciones como: accesorios, educación, juegos, oficina, entre otras que le ofrecen información al usuario. Además, está formado por más de 70 000 paquetes y actualizaciones disponibles en su repositorio y accesible mediante servicios web.

En la actualidad el proceso de migración a software libre en Cuba, se está desplegando mediante la transferencia de productos informáticos soportados en las tecnologías libres, en los diferentes Organismos de la Administración Central del Estado (OACE). El proceso de migración a Nova en los (OACE) ha estado asociado en los últimos años a la instalación en las instituciones de un repositorio de software. Cuando las computadoras son instaladas con Nova estos repositorios son configurados como orígenes para la instalación y actualización de software.

Durante el año 2017 fue instalado el repositorio nacional de aplicaciones de Nova, un servicio que permite a cualquier institución o usuario del país instalar software para Nova. Durante los despliegues en los organismos se han identificado a través de encuestas realizadas a los usuarios finales, por medio de las incidencias que llegan al Centro de Soporte, algunas aplicaciones de Nova que son usadas por la mayoría de los usuarios.

Los reportes mensuales emitidos por el Centro de Soporte a la UCI han permitido conocer cuáles son las aplicaciones que los consumidores por diversas cuestiones no pueden instalar desde los repositorios. A partir de los despliegues realizados y de la retroalimentación obtenida por las vías antes mencionadas, se ha podido conocer la insatisfacción de los usuarios finales con el uso de sistema operativo Nova.

De acuerdo a lo planteado previamente, se ha identificado como **problema de investigación** ¿Cómo brindar directamente la información asociada al uso del repositorio de Nova?

Para dar respuesta al problema científico expuesto se asume como **objeto de estudio:** proceso de obtención de información asociada al uso del repositorio de GNU/Linux, definiendo el **campo de acción:** proceso de obtención de información asociada al uso del repositorio de la distribución cubana de GNU/Linux Nova.

Para brindarle una solución efectiva al problema, se plantea como **objetivo general:** desarrollar una API REST que ofrezca la información asociada al uso del repositorio de la distribución cubana de GNU/Linux Nova.

A partir del objetivo general se determinan los **objetivos específicos** siguientes:

1. Elaborar el marco teórico de la investigación sobre el uso de los repositorios de sistemas operativos GNU/Linux.
2. Diseñar una API REST con la información asociada al uso del repositorio de la distribución cubana de GNU/Linux Nova.
3. Implementar una API REST con la información asociada al uso del repositorio de la distribución cubana de GNU/Linux Nova.
4. Evaluar la API REST desarrollada mediante pruebas de software y la técnica de Iadov.

Se definen como **preguntas científicas:**

1. ¿Cuáles son los presupuestos teóricos que fundamentan los procesos de obtención de información en el uso del repositorio de Nova?
2. ¿Qué aspectos se deben tener en cuenta para diseñar una API REST asociada al uso del repositorio de Nova?
3. ¿Cuáles son las herramientas y tecnologías más adecuadas para la implementación de una API REST asociada al uso del repositorio de Nova?
4. ¿Qué pruebas de software aplicar para la evaluación de la API REST asociada al uso del repositorio de Nova?

Para el cumplimiento del objetivo general planteado se utilizarán los siguientes métodos de investigación:

Métodos teóricos

- **Analítico - Sintético:** permitió descomponer el problema en sus partes para poder realizar un mejor análisis, tomando información de distintas fuentes bibliográficas y la extracción de los elementos más importantes que se relacionan con los procesos de obtención de información asociada al uso del repositorio de GNU/Linux.
- **Inductivo-Deductivo:** permitió integrar los métodos y funcionalidades, así como sus patrones de diseño para resolver problemas particulares de la solución en desarrollo, facilitando el análisis de los elementos generales a elementos más específicos.

Métodos Empíricos

- **Entrevista:** permitió obtener información respecto al uso del repositorio de la distribución cubana

de GNU/Linux Nova, así como requisitos funcionales y características de la propuesta de solución (anexo1).

- **Observación:** posibilitó conocer la estructura del repositorio de Nova, la agrupación de los paquetes, el ciclo de vida de un paquete, cómo se realizan las descargas de paquetes y la gestión de los *logs* del repositorio (anexo 2).
- **Encuesta:** se aplicó para evaluar la propuesta de solución desarrollada y conocer el índice de satisfacción de sus usuarios potenciales (anexo 4).

El presente trabajo está estructurado en introducción, tres capítulos, conclusiones, recomendaciones, referencias bibliográficas y anexos. La estructura capitular se describe a continuación:

Capítulo 1: Fundamentación teórica sobre el proceso de obtención de información asociada al uso del repositorio de GNU/Linux

En este capítulo se hace referencia a todos los elementos teóricos a tener en cuenta en la investigación del sistema a implementar, haciendo énfasis en los repositorios existentes. Además, se analizan mecanismos, técnicas, herramientas y lenguajes para darle solución a la problemática planteada.

Capítulo 2: Análisis y diseño de una API REST asociada al uso del repositorio de Nova

Se hace alusión a la solución propuesta, planteándose la forma en la que se va a desarrollar la misma, así como el trabajo con las herramientas que se utilizarán para su implementación. Además, se describe el proceso ágil basado en la metodología Variación AUP para la UCI.

Capítulo3: Implementación y evaluación de la API REST asociada al uso del repositorio de Nova

Este capítulo describe las tareas relacionadas con las disciplinas de implementación y pruebas de las funcionalidades, además la utilización de la técnica de Iadov empleada para verificar la satisfacción del cliente.

CAPÍTULO 1: Fundamentación teórica sobre el proceso de obtención de información asociada al uso del repositorio de GNU/Linux

Introducción

Con el objetivo de facilitar la comprensión del alcance de la investigación, en el presente capítulo se realiza un estudio sobre el proceso de obtención de información asociada al uso del repositorio de GNU/Linux. Por último, se caracterizan la metodología, tecnologías y herramientas utilizadas durante el ciclo de desarrollo de la solución que se propone.

1.1 Linux

Linux o GNU/Linux es un sistema operativo de software libre. Es multitarea, multiusuario, compatible con UNIX, proporciona una interfaz de comandos y una interfaz gráfica. El código fuente es accesible para que cualquier usuario pueda estudiarlo y modificarlo. Su licencia no restringe el derecho de venta, por lo que diversas empresas de software comercial distribuyen versiones de Linux. Cuenta con muchas distribuciones, gestores de ventanas para el entorno gráfico y cuentan con repositorios de código fuente y binarios (4) .

1.2 Distribución cubana GNU/Linux Nova

Nova es una distribución cubana basada en el sistema operativo GNU/Linux. Utiliza el núcleo de Linux e incluye determinados paquetes de aplicaciones informáticas para satisfacer las necesidades de la migración a plataformas de código abierto que experimenta Cuba como parte del proceso de informatización de la sociedad (5). Es el sistema operativo recomendado por El Grupo Nacional para la Migración a Aplicaciones de Código Abierto para ser utilizado como tecnología base de despliegue y desarrollo de aplicaciones informáticas en los Organismos de la Administración Central del Estado.

1.3 Repositorios de código fuente y binarios

Se define que un repositorio de código fuente y binarios es un sitio en la red donde se almacena el código fuente de las aplicaciones, así como binarios que son los que se instalan en el sistema operativo del usuario (2).

Teniendo en cuenta lo anterior:

- Un repositorio es un lugar en internet donde se almacena información, en el caso de los repositorios Linux esta información son programas.
- El repositorio es a todos los efectos un archivo ordenado donde son almacenados los paquetes Debian (sean estos paquetes binarios o fuente) en modo bien organizado, con una estructura bien definida y constantemente actualizados.

- Es un sitio centralizado donde se guarda información y archivos de los paquetes (imágenes, librerías, programas, código fuente) que se pueden instalar en Linux.

1.4 Paquete

Se define que un paquete es un archivo comprimido que contiene información del producto, archivos de programa, bibliotecas, íconos, documentación y scripts de configuración (2).

Paquete es un archivo comprimido que posee un conjunto de datos de aplicaciones informáticas que tiene una estructura definida, lo que hace que las herramientas de gestión de software del sistema puedan ejecutarlo para realizar compilaciones, instalaciones y eliminaciones de los archivos de configuración del sistema, también actualizaciones e instalaciones de nuevas aplicaciones y funcionalidades de una forma segura y centralizada (2).

La autora de la investigación después de analizar diferentes bibliografías considera que un paquete de software es un archivo comprimido con una estructura definida que contiene información necesaria de una aplicación para que pueda ser instalada, eliminada o actualizada en el sistema operativo de la máquina.

1.5 Gestores de paquetes

Los gestores de paquetes son herramientas que permiten el proceso de instalación, actualización o eliminación de paquetes de software desde los repositorios, adicionando funciones especiales como son: comprobación de firma digital, gestionar dependencias según se necesiten y comprobar la suma de verificación del paquete. Entre los gestores de paquetes que acepta Nova se encuentran dpkg, apt, aptitude y synaptic, utilizando en dicha cadena, la aparición de características mejoradas del anterior gestor; aunque también existen los estándares de repositorio de empaquetamiento entre los que se encuentran:

Metadatos: Según Howe (1993), el término fue acuñado por Jack Myers en la década de los 60 para describir conjuntos de datos. La primera acepción que se le dio (y actualmente la más extendida) fue (dato sobre el dato), ya que proporcionaban la información mínima necesaria para identificar un recurso que puede incluir información descriptiva sobre el contexto, calidad y condición o características del dato (6).

Estos paquetes contienen información adicional, comúnmente conocida como metadatos, tales como (7):

- un resumen del propósito general del software,
- una descripción,

- una lista de archivos contenidos en el paquete,
- la versión del software que contiene, así como el número de versión del paquete,
- cuándo, dónde y por quién se ha construido,
- la arquitectura para la que se ha construido,
- comprobación de los archivos contenidos en el paquete (suma de verificación),
- la licencia del software que contiene,
- qué otros paquetes necesitan para funcionar correctamente (dependencias),

Paquetes de dependencia:

Los paquetes de software y los paquetes de dependencias son aspectos muy importantes de las distribuciones de GNU/Linux ya que proporcionan una forma modular para configurar y administrar un sistema operativo y sus aplicaciones. Un aspecto importante del archivado de los paquetes son las interacciones que contienen:

- Los paquetes de dependencias son transitivos, lo que significa que cuando un paquete A necesita el paquete B, y el paquete B necesita el paquete C, significa que el paquete A también necesita el paquete C, por lo que a veces se termina instalando un montón de paquetes extras a pesar de que simplemente quería una sola aplicación.

- Las bibliotecas de dependencias (por lo general los paquetes con un nombre que comienza con "lib" son muy comunes y casi cada aplicación depende de un conjunto de paquetes de bibliotecas.

La información contenida en los repositorios de las distribuciones de GNU/Linux, es brindada a los usuarios mediante acceso remoto y servicios web.

1.6 Servicios web que brindan información sobre los repositorios de distribuciones GNU/Linux

En la actualidad existen múltiples definiciones sobre los servicios web como, una colección de procedimientos (métodos) a los que se puede acceder desde cualquier lugar de internet, siendo este un mecanismo de invocación totalmente independiente de la plataforma que se utilice y del lenguaje de programación en el que se haya implementado el servicio.

El WC3 (*World Wide Web Consortium*) lo define como un software diseñado para soportar interacciones máquina a máquina a través de la red. Proporcionan una forma estándar de interoperar aplicaciones de software que se ejecutan en diferentes plataformas. Su principal característica es su gran interoperabilidad y extensibilidad, así como proporcionar información fácilmente procesable por las máquinas (8).

Teniendo en cuenta lo anteriormente planteado un servicio web es un servicio ofrecido por una aplicación al cliente desde cualquier plataforma mediante una interfaz accesible a través de la red. Con el objetivo de hacer un estudio de los sistemas homólogos que ofrecen información de acceso a los repositorios se realizó un estudio bibliográfico de diferentes fuentes en la búsqueda de información relacionada con este aspecto. Fueron realizadas búsquedas en internet siguiendo criterios relacionados con los términos "*web service, api, access, repository, logs, ubuntu, suse, redhat, fedora*" y varias combinaciones entre ellos que permitieran acceder a información sobre cómo las distintas distribuciones ofrecen datos del acceso a sus repositorios.

Las búsquedas en internet no proporcionaron resultados con información relevante para el desarrollo de la investigación, a partir de la ausencia de resultados se procedió a hacer un estudio al Centro de Software de Ubuntu debido a que es una herramienta que ofrece información acerca de las aplicaciones contenidas en el repositorio, algo que el Centro de Software de Nova no provee en su última versión. El mismo ofrece información acerca de las aplicaciones alojadas en el repositorio, clasifica las aplicaciones por categorías, registra las aplicaciones favoritas y otras tantas secciones. En el estudio se realizó una entrevista a los desarrolladores del proyecto de Nova (anexo 1), para poder determinar qué información era necesaria conocer sobre los paquetes que se alojan en el repositorio. Se llegó a la conclusión que se necesita una API REST que disponga información acerca del repositorio teniendo en cuenta:

- Aplicaciones más utilizadas.
- Aplicaciones favoritas.
- Aplicaciones por categorías
- Aplicaciones que muestren su éxito de descargas.
- Aplicaciones que muestren fallo en sus descargas.
- La fecha de las aplicaciones más descargadas.
- Obtener el IP de donde son realizadas las descargas.
- Obtener un ranking de los fallos que realicen las descargas.
- Sección para las sugerencias de los usuarios.

1.7 API REST

Las siglas API vienen del inglés (*Application Programming Interface,* Interfaz de Programación de Aplicaciones), es un conjunto de funciones y procedimientos que cumplen una o muchas funciones con el fin de ser utilizadas por otro *software* (9). La API permite implementar las funciones y procedimientos que engloba en el proyecto sin la necesidad de programarlas de nuevo. En términos de programación, es una capa de abstracción**.**

REST *(Representational State Transfer,* Transferencia de Estado Representacional*)* es cualquier interfaz entre sistemas que usen HTTP para obtener datos o generar operaciones sobre esos datos en todos los formatos posibles, como XML y JSON (10).

REST se define como un tipo de arquitectura más natural y estándar para crear APIs para servicios orientados a Internet. Es un tipo de arquitectura de desarrollo web que se apoya totalmente en el estándar HTTP. REST permite crear servicios y aplicaciones que pueden ser usadas por cualquier dispositivo o cliente que entienda HTTP, por lo que es increíblemente más simple y convencional que otras alternativas que se han usado en los últimos años (11).

Características de REST:

- Protocolo cliente/servidor sin estado: cada petición HTTP contiene toda la información necesaria para ejecutarla, lo que permite que ni cliente ni servidor necesiten recordar ningún estado previo para satisfacerla.
- Las operaciones más importantes relacionadas con los datos en cualquier sistema REST son:

  POST: crear un recurso
  GET: leer y consultar un recurso
  PUT: editar un recurso
  DELETE: eliminar un recurso
- Los objetos en REST siempre se manipulan a partir de la URL, el identificador único de cada recurso de ese sistema REST. Facilita acceder a la información para su modificación, borrado o para compartir su ubicación exacta con terceros.
- Interfaz uniforme: para la transferencia de datos en un sistema REST, este aplica acciones concretas (POST, GET, PUT y DELETE) sobre los recursos, siempre y cuando estén identificados con una URL. Esto facilita la existencia de una interfaz uniforme que sistematiza el proceso con la información.
- Sistema de capas: arquitectura jerárquica entre los componentes. Cada una de estas capas lleva a cabo una funcionalidad dentro del sistema REST.
- Uso de hipermedios: En el caso de una API REST, el concepto de hipermedia explica la capacidad de una interfaz de desarrollo de aplicaciones de proporcionar al cliente y al usuario los enlaces adecuados para ejecutar acciones concretas sobre los datos.

Ventajas que ofrece REST para el desarrollo (10).

- Separación entre el cliente y el servidor: el protocolo REST separa totalmente la interfaz de usuario del servidor y el almacenamiento de datos. Eso tiene algunas ventajas cuando se hacen desarrollos. Por ejemplo, mejora la portabilidad de la interfaz a otro tipo de plataformas, aumenta la escalabilidad de los proyectos y permite que los distintos componentes de los desarrollos se puedan evolucionar de forma independiente.
- Visibilidad, fiabilidad y escalabilidad. La separación entre cliente y servidor tiene una ventaja evidente y es que cualquier equipo de desarrollo puede escalar el producto sin excesivos problemas. Se puede migrar a otros servidores o realizar todo tipo de cambios en la base de datos, siempre y cuando los datos de cada una de las peticiones se envíen de forma correcta. Esta separación facilita tener en servidores distintos el *front* y el *back* y eso convierte a las aplicaciones en productos más flexibles a la hora de trabajar.
- La API REST siempre es independiente del tipo de plataforma o lenguaje: la API REST siempre se adapta al tipo de sintaxis o plataformas con las que se estén trabajando, lo que ofrece una gran libertad a la hora de cambiar o probar nuevos entornos dentro del desarrollo. Con una API REST se pueden tener servidores *PHP, Java, Python o Node.js.* Lo único que es indispensable es que las respuestas a las peticiones se hagan siempre en el lenguaje de intercambio de información usado, normalmente XML o JSON.

La autora de la investigación después de analizar diferentes bibliografías considera que una API REST ofrece tanto al cliente como al servidor, las interacciones mediante construcciones que son manejables por el usuario debido a su conocimiento al interactuar con el Protocolo de Transferencia de Hipertexto (HTTP), teniendo presente sus principios básicos, además de la interacción con otras aplicaciones. A continuación, se describen la metodología, herramientas y tecnologías empleadas para desarrollar la API REST que se propone como propuesta de solución en la presente investigación.

1.8 Metodología de desarrollo de software

Una metodología de desarrollo de software es un marco de trabajo usado para estructurar, planificar y controlar el proceso de desarrollo de un sistema informático. Es un conjunto de procedimientos, técnicas, artefactos y herramientas que guían a los desarrolladores en la realización de software (12). La metodología de desarrollo de software Variación de AUP para la UCI es la utilizada para la elaboración de la propuesta de solución.

1.8.1 Metodología Variación de AUP para la UCI

La metodología Variación de AUP para la UCI es una variante de la metodología AUP (*Agile Unfied Process*, Proceso Unificado Ágil). La misma fue definida por la universidad para guiar el proceso de desarrollo de software en la institución ya que se adapta al ciclo de vida definido para la actividad productiva, se ajusta a las características de cada proyecto y lograr una mayor homogeneidad entre los procesos de desarrollo de todos los centros productivos de la UCI (12).

Esta metodología contiene tres fases: Inicio, Ejecución y Cierre. El desarrollo de la propuesta de solución se centra en la fase de Ejecución ya que es donde se desarrollan las actividades para especificar los requisitos de software, diseñar, implementar y probar la solución.

En la elaboración de la herramienta informática propuesta, se emplea el escenario 4 de metodología, para la descripción de los requisitos ya que se aplica a los proyectos que tienen el negocio bien definido, el cliente acompaña al equipo de desarrollo para convenir los detalles de los requisitos y así poder implementarlos, probarlos y validarlos y el proyecto no es extenso.

1.9 Lenguaje y herramienta para el modelado de la solución

En el modelado de la propuesta de solución se utilizó el siguiente lenguaje y herramienta, descritos a continuación:

1.9.1 Lenguaje Unificado de Modelado (UML)

El Lenguaje Unificado de Modelado (UML) es el lenguaje para especificar, visualizar, construir y documentar los artefactos de los sistemas de software, así como para el modelado del negocio. Posee vocabularios y reglas que se centran en la representación conceptual y física de un sistema, para crear y leer modelos bien formados que son una de las características que posee este lenguaje, además se basa en una notación gráfica y concisa; y es un lenguaje de modelado visual de propósito general orientado a objetos (13).

UML es utilizado en la presente investigación para modelar los diferentes productos de trabajo que se obtienen en las disciplinas de Análisis y diseño e Implementación, y la obtención de Requisitos necesarios para el desarrollo de la propuesta de solución.

1.9.2 Herramienta Visual Paradigm 8.0 para UML

Visual Paradigm es una herramienta CASE *(Computer Aided Software Engineering,* Ingeniería de Software Asistida por Computadoras) que emplea UML como lenguaje de modelado. Propicia un conjunto de ayudas para el desarrollo de programas informáticos a través de la representación de todo tipo de diagramas. Está concebido para soportar el ciclo de vida completo del proceso de desarrollo de

software. Ofrece un lenguaje estándar común a todo el equipo de desarrollo, lo cual facilita considerablemente la comunicación. Permite la generación de código fuente y bases de datos mediante la transformación de diagramas de Entidad-Relación en tablas de base de datos. Posee licencia gratuita y se puede utilizar en múltiples plataformas (13).

1.10 Tecnologías de implementación

En el desarrollo de la propuesta de solución se utilizaron las siguientes tecnologías:

1.10.1 Marco de trabajo

Un marco de trabajo o *framework* es un diseño abstracto orientado a objetos para un determinado tipo de aplicación, es un patrón arquitectónico que proporciona una plantilla extensible para un tipo específico de aplicaciones. Un marco de trabajo es un subsistema expandible de un conjunto de servicios, es un conjunto cohesivo de interfaces y clases que colaboran para proporcionar los servicios de la parte central e invariable de un subsistema lógico (14).

Flask

Flask es un marco de trabajo ligero de aplicaciones web. Está diseñado para que el inicio sea rápido y fácil, con la capacidad de escalar hasta aplicaciones complejas. Comenzó como un simple envoltorio alrededor de *Werkzeug* y Jinja y se ha convertido en uno de los *frameworks* de aplicación web Python más populares. *Flask* ofrece sugerencias, pero no impone ninguna dependencia ni diseño del proyecto. Depende del desarrollador elegir las herramientas y bibliotecas que quiere usar. Hay muchas extensiones proporcionadas por la comunidad que facilitan la incorporación de nueva funcionalidad (15).

La elección del *framework* a desarrollar en la propuesta de solución, se escogió a *Flask* por ser un *microframework* orientado a desarrollar API REST por sus características antes mencionadas.

1.10.2 Lenguaje de programación

Los lenguajes de programación permiten crear software. Están diseñados para ejecutar un conjunto de acciones consecutivas que un equipo debe ejecutar. Además, se utilizan principalmente para controlar cómo se comporta una máquina o crear programas informáticos (16). A continuación, se describen los lenguajes de programación definidos para el desarrollo de la solución.

Python 3.5

Es un lenguaje interpretado de programación, multiplataforma, flexible, lenguaje potente, de sintaxis clara y concisa, su implementación está bajo la licencia de código abierto *Python Software Foundation License* (Licencia permisiva de la Fundación de Software Python). Es un lenguaje interpretado e implica ahorro de tiempo durante el desarrollo de un programa ya que no necesita de compilación. El intérprete

puede usarse de modo interactivo, el cual hace fácil el experimentar con las características del lenguaje para probar funciones durante la etapa inicial de desarrollo (17).

Las principales características que presenta este lenguaje son (18):

- Programación orientada a objetos.
- Tiene tipado dinámico.
- Es un lenguaje interpretado.
- Es un lenguaje multiplataforma.
- Puede utilizarse como un lenguaje de extensión para módulos y aplicaciones que necesitan de una interfaz programable.

La elección de Python para el desarrollo de la propuesta de solución está fuertemente relacionada con la necesidad de implementar una API REST con *Flask*[3], que está escrito en Python. Existen otros frameworks para la creación de API REST, basados en diferentes lenguajes de programación. Python ofrece soporte suficiente para las tecnologías que, a priori, se consideran necesarias para llevar a cabo el proyecto, como JSON o Elasticsearch.

1.10.3 Entorno de Desarrollo Integrado

Un entorno de desarrollo integrado (IDE, *Integrated Development Environment*) es un programa compuesto por un conjunto de herramientas para que el programador las utilice. Puede contener un solo lenguaje de programación o bien puede utilizarse para varios. El entorno de desarrollo es imprescindible en la producción de un software (19).

PyCharm 3.2: Es un IDE multiplataforma utilizado para desarrollar en el lenguaje de programación Python. Proporciona análisis de código, depuración gráfica y soporte para el desarrollo web con *Flask*. La empresa desarrolla dos versiones de licencias: la *Community* que es gratuita y orientada a la educación y al desarrollo puro en Python y la Professional, que incluye más características como el soporte a desarrollo web. Admite lenguajes como JavaScript, Angular, entre otras. Ofrece gran compatibilidad con *frameworks* (marcos de trabajo) de desarrollo web modernos como: *Django, Flask, Google App* (20).

[3] Flask: Microframework orientado a la creación de APIs

1.10.4 Elasticsearch

A diferencia de otros motores de búsqueda y recuperación de información, *Elasticsearch* actúa como repositorio de información, almacenando los documentos que indexa. Esto permite reemplazar almacenes de documentos como *MongoDB* o *Raven DB* por *Elasticsearch* en los proyectos, y se considera necesario. Tanto *MongoDB* como *Elasticsearch* son repositorios de documentos desnormalizados en los que los documentos que gestionan no necesitan disponer de un esquema rígido para poderlos introducir en el sistema (21).

Las Bases de Datos relacionales (SQL) requieren definir las tablas, o esquema de la información, antes de poder introducir registros de datos. Esto hace que se tenga que pensar muy bien la estructura de la información antes de comenzar a programar en un proyecto para evitar que se produzcan cambios, puesto que estos cambios representarían una migración de los datos a otro esquema o tenerlos que adaptar. Si se dispone de un volumen de datos muy grande, se estaría hablando de un proceso muy lento y costoso (21).

Sin embargo, como *Elasticsearch* no requiere un esquema predefinido de los datos, la misma colección de documentos puede contener documentos de estructura diferente, permitiendo un desarrollo más ágil. Por otro lado, los documentos almacenados en *Elasticsearch* están en formato *JSON* que son estructuras de datos más cercanas a las entidades que se manejan a nivel de negocio. Esto también facilita un desarrollo más ágil ya que el propio documento contiene todos los elementos de la entidad de negocio sin tener la necesidad de lanzar consultas complejas sobre una base de datos para obtenerlos y poderlos procesar posteriormente. Por otro lado, mediante su RESTful API este motor de búsqueda será ideal para realizar cualquier integración con otras tecnologías (21).

El surgimiento de las bases de datos no relacionales (NoSQL) dio lugar a una serie de sistemas de gestión de base de datos, con un enfoque al rendimiento, la fiabilidad y la coherencia, para satisfacer las necesidades a grandes volúmenes de información. Teniendo en cuenta las necesidades antes planteadas una de las principales ventajas que poseen las bases de datos no relacionales sobre la base de datos relacionales (SQL), es que presentan un almacenamiento no estructurado. Cuando se interactúa con las bases de datos NoSQL, ya sean de código abierto o tengan un propietario, la expansión es más fácil y más barata que cuando se trabaja con bases de datos relacionales. Esto se debe a que se realiza un escalado horizontal y se distribuye la carga por todos los nodos, en lugar de realizarse una escala vertical, más típica en los sistemas de bases de datos relacionales (22).

Elasticsearch (ES) es un motor de búsqueda orientado a documentos *JSON* estructurados sin schemas, desarrollado en *Java* y de código abierto, una de las principales características es que permite tener una

arquitectura distribuida, escalable y de alta disponibilidad, ES se basa en Lucene para las búsquedas de texto. Las búsquedas soportan multi-idioma, geolocalización, autocompletado, sugerencias; ideal para los proyectos en donde se trabaja con big data (21).

La elección de *Elasticsearch* para la propuesta de solución se debe principalmente a varios factores, el uso de *JSON* como sistema de escritura de los datos, la potencia para la búsqueda de estos datos una vez indexados, además por las consultas DSL en la *query.* Teniendo en cuenta sus principales características y el manejo de grandes volúmenes de información que requiere la presente investigación

A continuación se expondrán varios conceptos que son fundamentales en el estudio de *Elasticsearch* (23):

Índice: Un índice es un espacio de nombres lógico mapeado en uno o más fragmentos primarios los cuales pueden tener ninguna o varias réplicas. Es una colección de documentos que tienen características similares. Los índices están identificados por un nombre, el cual se utilizará a la hora de indexar, buscar, actualizar y borrar. Indexar un documento es crear un documento para que pueda ser buscado.

El índice es el lugar (lógico) donde *Elasticsearch* almacena los datos de modo que se pueden dividir a su vez en trozos más pequeños denominados fragmentos. Sería como una tabla en el mundo de las bases de datos. La estructura del índice se prepara para una rápida y eficiente búsqueda de texto completo, pero sin almacenar los valores originales. *Elasticsearch* puede contener muchos índices situados en una máquina o repartidos en varios nodos. Cada índice es construido de uno o más fragmentos *(shards),* y cada fragmento puede tener muchas réplicas.

Documento: Usando la analogía de las bases de datos relacionales, un documento es una tabla, normalmente es un documento *JSON*. Los documentos se dividen a su vez en campos *(fields*), y puede estar varias veces en un mismo documento (*multivalued*). Cada campo tiene un tipo de dato (texto, número, fecha, etc.), los cuales pueden ser también complejos y albergar otros subdocumentos o tablas.

Afortunadamente la asignación a tipo de dato se puede determinar automáticamente (sin embargo, puede ser recomendable utilizar asignaciones). Un documento en *Elasticsearch* debe tener el mismo tipo de dato para todos los campos comunes.

Los documentos no necesitan tener una estructura fija, pudiendo tener un conjunto de campos diferentes. Los campos no tienen por qué que ser conocidos durante el desarrollo de la aplicación. Desde el punto de vista del cliente, un documento es un objeto *JSON*.

Cada documento está almacenado en un índice y tiene su propio identificador único (que puede ser generado automáticamente por *Elasticsearch*) y tipo de documento. Un documento debe tener un identificador único en relación con su tipo. Esto significa que, en un solo índice, dos documentos pueden tener el mismo identificador único si no son del mismo tipo de documento.

Tipo de documento: En *Elasticsearch*, un índice puede almacenar muchos objetos con diferentes propósitos. El tipo de documento permite diferenciar fácilmente entre los objetos en un índice determinado a la vez que facilita la manipulación de datos.

Mapping (Asignaciones)**:** En cuanto a la preparación de un texto para ser indexado y posteriormente buscado, como se ha dicho anteriormente, cada campo del documento debe ser analizado correctamente en función de su tipo. Por ejemplo, se requieren procesos de análisis diferentes para los campos numéricos que para los de texto, los números no pueden ser ordenados alfabéticamente. *Elasticsearch* almacena la información sobre los campos dentro del *mapping*.

Conceptos claves en Elasticsearch

Desde el principio*, Elasticsearch* fue creado como una solución distribuida que puede manejar miles de millones de documentos y cientos de solicitudes de búsqueda por segundo. Esto se debe a varios conceptos importantes que se van a describir con más detalle a continuación.

Nodo y Clúster: *Elasticsearch* puede trabajar de forma autónoma en una sola instancia de servidor. Sin embargo, para procesar grandes conjuntos de datos, lograr tolerancia a fallos y alta disponibilidad*, Elasticsearch* se pueden ejecutar en muchos servidores y funcionar de forma cooperativa en clúster, denominando a cada servidor de la formación nodo.

Shard (Fragmento)**:** Cuando se tiene un gran número de documentos, se puede llegar a un punto en el que un único nodo puede no ser suficiente, debido a limitaciones de memoria RAM, capacidad de disco duro, potencia de procesamiento o simplemente la incapacidad para responder a las solicitudes de clientes lo suficientemente rápido. En tal caso, los documentos se pueden dividir en partes más pequeñas llamadas fragmentos (donde cada fragmento es un índice de *Apache Lucene* separado).

Cada fragmento se puede colocar en un servidor diferente, y, por lo tanto, sus datos se pueden transmitir entre los nodos del clúster. Cuando se consulta un índice dividido en varios fragmentos, *Elasticsearch* envía la consulta a cada fragmento relevante y fusiona el resultado acelerando por tanto la indexación, como es de suponer ese proceso es totalmente transparente para el cliente. Durante una indexación es posible especificar cuantos *shards* y réplicas deben crearse.

Réplica: Con el fin de aumentar el rendimiento de consulta o lograr una alta disponibilidad, se pueden

usar réplicas de fragmento. Una réplica es sólo una copia exacta de un *shard*, y cada fragmento puede tener cero o más réplicas. *Elasticsearch* puede tener muchos fragmentos idénticos y uno de ellos es elegido a la hora de tener que realizar cambios sobre el índice. Este fragmento especial se llama un fragmento primario, y los demás se llaman fragmentos de réplica. Cuando se pierde el fragmento primario (por ejemplo, un servidor que contiene el fragmento de datos no está disponible), el clúster cambia el estado de un fragmento de réplica para ser el nuevo fragmento primario.

Sharding es importante por dos razones principales: Permite dividir horizontalmente o reducir el volumen de contenido. Permite distribuir y paralelizar las operaciones a través de fragmentos (potencialmente en varios nodos) aumentando así el rendimiento.

La replicación es importante por dos razones principales: Proporciona alta disponibilidad en caso de que falle un fragmento/nodo. Por esta razón, es importante tener en cuenta que un fragmento de réplica no se asigna en el mismo nodo que el fragmento primario/original que fue copiado. Se le permite escalar volumen de búsquedas y rendimiento puesto que las búsquedas se pueden ejecutar en todas las réplicas en paralelo.

Características de *Elasticsearch*:

- Orientado a documentos.
- *JSON*, basado en *Apache Lucene.*
- Libre de *Schema.*
- Presenta una arquitectura; distribuida, escalable en alta disponibilidad.
- Capacidades de búsqueda y análisis en tiempo real mediante peticiones por el método GET.
- Permite consulta sobre uno o varios índices.
- Permite múltiples índices en un *clúster*, así como alias de índice.
- *Full Text Search* o búsqueda por texto completo.
- Indexa todos los campos de los documentos *JSON* sin esquemas rígidos.
- Búsquedas mediante *Query DSL (Domain Specific Language).*
- Escala de forma dinámica.
- Centrado en APIs.
- Expone sus funcionalidades vía API REST.

ES presenta dos tipos de búsqueda, la básica y la compleja, a continuación, se ofrecerá información:

Búsquedas básicas: ES implementa poderosos mecanismos para las búsquedas. Las búsquedas son posibles a través de *Search* API:

- Relevancia (*Scoring*): Define qué tan importante es un documento en un conjunto de resultados.
- *Spellchecker*: Permite interpretar una búsqueda, aunque tenga errores ortográficos.
- Soporte multi-lenguaje: Buscar en varios lenguajes.
- *Autocomplete*: Predice búsquedas con autocompletado.

Búsquedas complejas (DSL): ES implementa poderosos mecanismos para las búsquedas. Las búsquedas son posibles a través de *Search API* y del lenguaje DSL.

- *Query* DSL permite: Realizar búsquedas complejas en documentos. Se basa en la composición de querys.
- *Query*: Consulta realizada.
- *Filter*: Filtro aplicado sobre la query.

Dándole complejidad a las búsquedas:

- Búsquedas por rangos: Para buscar en un rango específico, aplicable a valores numéricos y fechas. (Implementa los operadores lte, gte, lt, gt).
- Búsquedas por términos: Empleada para obtener valores exactos.
- Búsqueda por existencia o no existencia: Para obtener documentos donde exista un campo determinado (llave del *JSON*).
- Devolver todos los documentos: Se obtiene empleando *match_all*.
- Búsqueda difusa: Similar a una búsqueda normal, con la diferencia de que la búsqueda difusa compara y ordena todos los valores de campo según su grado de parecido con la cadena de búsqueda introducida.

1.6.5 Fluentd

Es un recolector que unifica *logs*, proyecto *open source* que permite aunar colecciones de datos y que sean consumidos de forma centralizada para poder entender mejor los datos. La combinación de *fluentd* + *elasticsearch* + *elasticsearchHQ* permite unificar, almacenar y visualizar todos los *logs* de todos los contenedores en el equipo linux en un único sitio. Al combinar las tres herramientas *(Fluentd* + *Elasticsearch*+ *ElasticsearchHQ)* se consigue un sistema escalable, sencillo y flexible que agrega los logs en un motor de búsqueda que puede ser consumido de forma sencilla desde la web mediante una API REST (24).

JSON

JSON es el acrónimo de *Java Script Object Notation*, es un formato de texto ligero para intercambiar y gestionar datos. Teniendo en cuenta las distintas tecnologías que intervienen en el proyecto, el uso de JSON se hace casi necesario, sobre todo en el uso de las base de datos no relacionales como *Elasticsearch* y *Mongo DB,* o algunas API que devuelven las consultas que se le hacen en este tipo de formato, agregándole que es fácilmente integrable con el resto de las tecnologías del proyecto (18). Otra razón por la cual se escoge este formato es por el potente soporte que ofrece, almacenada en la librería estándar de *Python*.

1.6.6 Servidor de aplicaciones web

Los servidores web (también conocidos como servidores HTTP) son un tipo de servidores utilizados para la distribución de contenido web en redes internas o en Internet. Como parte de una red de ordenadores, un servidor web transfiere documentos a los llamados clientes. Los servidores web sirven para almacenar contenidos de Internet y facilitar su disponibilidad de forma constante y segura.

Además, un servidor web permite el cifrado de la comunicación entre el servidor web y el cliente, autenticación HTTP para áreas específicas de una aplicación web y almacenamiento en caché de documentos dinámicos para la respuesta eficiente de solicitudes y evitar una sobrecarga del servidor (Servidor de aplicaciones -IBM). En el desarrollo de la herramienta para la creación de Servicios web asociado al repositorio de Nova se utilizará como servidor de aplicaciones web Apache.

Apache

Es un servidor de red para el protocolo HTTP, elegido para poder funcionar como un proceso independiente, sin que eso solicite el apoyo de otras aplicaciones o directamente del usuario. *Apache* se distribuye como software libre de código abierto, modular, multiplataforma, extensible, popular (fácil de conseguir ayuda/soporte) y gratuito. Permite su uso comercial y no comercial (2).

Características:

- Altamente configurable, de diseño modular y permite la creación y gestión de registros (*logs*).
- Personaliza la respuesta ante los posibles errores que puedan ocurrir.

Se selecciona el servidor web *Apache*, debido a que es el utilizado en la aplicación a la que se debe integrar la presente propuesta de solución, además de la gran gama de características que evidencia. Se utiliza la versión 2.2.

Conclusiones parciales

En el presente capítulo se hace un análisis exhaustivo sobre el proceso de obtención de información asociada al uso del repositorio de GNU/Linux. El análisis del Centro de Software de Ubuntu, la entrevista realizada a los desarrolladores del proyecto de Nova; permitieron definir los aspectos más importantes a tener en cuenta para realizar la API REST que se propone implementar. El estudio permitió establecer pautas necesarias para su utilización en la propuesta de solución teniendo en cuenta sus elementos característicos.

CAPÍTULO 2: Análisis y diseño de una API REST asociada al uso del repositorio de Nova

Introducción

El presente capítulo tiene como objetivo fundamental describir la propuesta de solución y las tareas de ingeniería realizadas y los productos de trabajo obtenidos durante la ejecución de las disciplinas de Requisitos, Análisis y diseño.

2.1 Descripción de la propuesta de solución

En la descripción de la propuesta de solución se elaboró un modelo conceptual. Este modelo permitió identificar los conceptos existentes en el proceso a informatizar, así como sus atributos y relaciones, lo que permitió conocer las características del entorno en el que se desarrolla la solución.

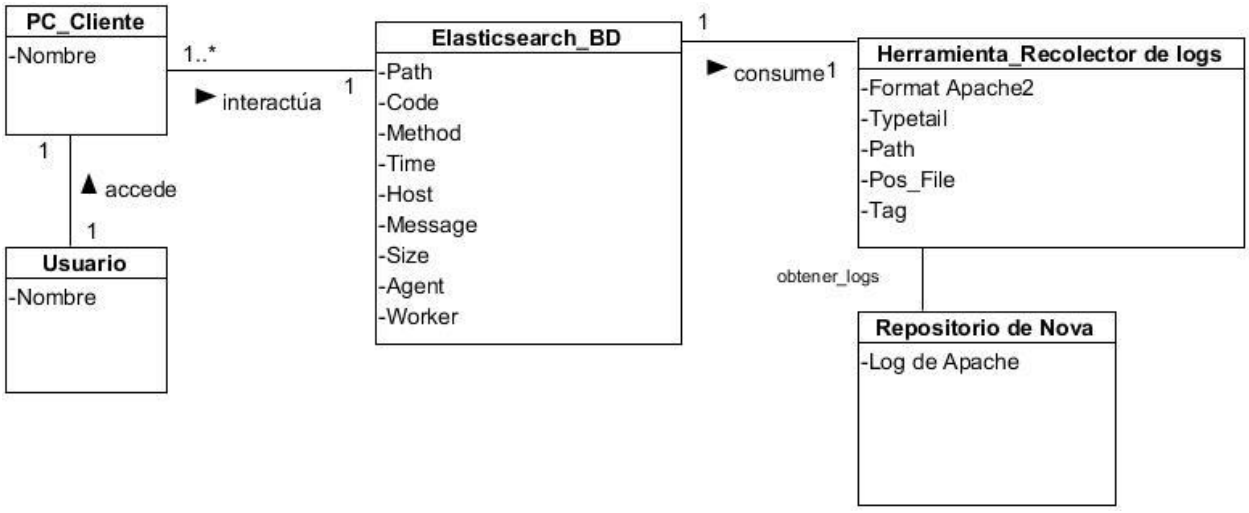


Figura 1. Modelo conceptual

(Fuente: Elaboración propia)

A continuación, se detallan los diferentes objetos y relaciones que conforman el proceso:

Repositorio de Nova: se refiere a los repositorios de código fuente y binarios, está compuesto por los paquetes del sistema, o sea los logs de Apache.

Elasticsearch_BD: se refiere a la base de datos que contiene toda la información de los logs de Apache que tiene el repositorio.

Usuario: es la persona que trabaja con la herramienta y que su interacción con el sistema depende de sus responsabilidades dentro del negocio.

PC_Cliente: es la máquina que interactúa con la herramienta y a su vez con el usuario.

Herramienta_Recolector de logs: es la herramienta encargada de conectar los *logs* del repositorio a la base de datos en tiempo real.

Interpretadas las características de la propuesta de solución, se desarrolló la disciplina de Requisitos de la metodología AUP para la UCI empleada en la investigación. En el epígrafe 2.2 se describen las tareas realizadas en esta disciplina, así como los productos de trabajo obtenidos.

2.2 Requisitos

El esfuerzo principal en la disciplina Requisitos es desarrollar un modelo del sistema que se va a construir. Esta disciplina comprende la administración y gestión de los requisitos del producto (12).

Los requisitos para un sistema son la descripción de los servicios proporcionados por el sistema y sus restricciones operativas. Estos requisitos reflejan las necesidades de los clientes de un software, esto ayuda a que exista una mayor comunicación entre el cliente y el equipo de desarrollo. Existen dos tipos de requisitos: los requisitos funcionales y los no funcionales (25). A continuación, se describe cómo se obtuvieron los requisitos de la propuesta de solución, su especificación, descripción y validación.

2.2.1 Obtención de requisitos

Las técnicas de identificación de requisitos de software permiten identificar las necesidades de negocio de los clientes y los usuarios. Son mecanismos que se utilizan para recolectar la información necesaria en la obtención de los requisitos de una aplicación, permiten investigar aspectos generales para posteriormente ser especificados con un mayor detalle, requieren ser adecuadamente orientadas para cubrir la información que se requiere capturar (26). A continuación, se describe la obtención de requisitos de la propuesta de solución mediante la entrevista y la observación.

Entrevista

La entrevista es de gran utilidad para obtener información cualitativa como opiniones, o descripciones subjetivas de actividades. Es una técnica muy utilizada y requiere una mayor preparación y experiencia por parte del analista. La entrevista se puede definir como un “intento sistemático de recoger información de otra persona” a través de una comunicación interpersonal que se lleva a cabo por medio de una conversación estructurada (27).

La entrevista (anexo1) se realizó a 5 especialistas de CESOL, que trabajan directamente en el mantenimiento y control del repositorio de Nova. La misma permitió obtener información sobre el uso del repositorio de la distribución cubana GNU/Linux Nova, así como requisitos funcionales y no funcionales de la propuesta de solución.

Observación

La técnica de observación facilita la obtención de información sobre la forma en que se efectúan las actividades. Permite observar la manera en que se llevan a cabo los procesos en una organización y verificar que realmente se sigan todos los pasos especificados en su funcionamiento (27).

La observación realizada (anexo 2) al proceso de mantenimiento y control del repositorio de Nova le permitió a la autora de la investigación conocer: la estructura del repositorio de Nova, la agrupación de los paquetes, el ciclo de vida de un paquete, cómo se realizan las descargas de paquetes y la gestión de los logs del repositorio.

2.2.2 Especificación de requisitos de software

Una especificación de requerimientos de software (ERS) es un documento que se crea cuando debe especificarse una descripción detallada de todos los aspectos del software que se va a elaborar (26). A continuación, se describen los requisitos funcionales y no funcionales de la propuesta de solución.

Requisitos funcionales

Los requisitos funcionales (RF) son declaraciones de los servicios que debe proporcionar el sistema, describe lo que este debe hacer. En la tabla 1 se especifican los requisitos funcionales que debe cumplir la propuesta de solución. La complejidad de los requisitos funcionales se obtuvo mediante el producto de trabajo Evaluación de requisitos, propuesto en el expediente de proyecto 5.0 para la actividad productiva de la universidad.

Tabla 1. Listado de los requisitos funcionales

(Fuente: Elaboración propia)

Número	Requisito funcional	Descripción	Complejidad
RF.1	Mostrar listado de paquetes	Permite mostrar un listado de los paquetes existentes en la base de datos y la categoría asociada a cada uno de ellos, en caso de que tenga una categoría asociada	Alta
RF.2	Mostrar listado de los paquetes más populares	Permite mostrar un listado con los paquetes más populares que se define por la cantidad de veces que se ha descargado ese paquete.	Alta

RF.3	Mostrar la cantidad de veces que se descarga un paquete con éxito	Permite mostrar la cantidad de veces que se descarga un paquete con éxito, además del nombre del paquete.	Alta
RF.4	Mostrar la cantidad de veces que un paquete tiene descarga fallida	Lo que muestra es un número o un listado con la cantidad de veces que ha fallado cada paquete, en dependencia de la cantidad introducida por el usuario.	Alta
RF.5	Mostrar los paquetes que más fallan	Permite mostrarle al usuario el número de descargas fallidas.	Alta
RF.6	Mostrar dirección hacia donde se descargan las paquetes	Permite mostrarle al usuario las direcciones IP donde son descargados los paquetes.	Media
RF.7	Mostrar los paquetes más descargadas en una fecha	Permite mostrarle al usuario los paquetes descargados dada una fecha.	Media
RF.8	Autenticar usuarios	Permite autenticar a los usuarios que consuman del servicio.	Alta

Requisitos no funcionales

Los requisitos no funcionales (RNF) son restricciones de los servicios y cualidades ofrecidas por el sistema que surgen en función de las necesidades del usuario (26). En la tabla 2 se especifican los requisitos no funcionales de la propuesta de solución. Los mismos hacen referencia a las características o atributos de calidad propuestos en el producto de trabajo Especificación de requisitos de software del expediente de proyecto 5.0 para la actividad productiva de la universidad.

Tabla 2. Listado de los requisitos no funcionales

(Fuente: Elaboración propia)

Número	Requisito no funcional	Descripción
RNF 1	Usabilidad	El sistema está creado para ser utilizado por personas con conocimientos básicos de informática y el idioma de la interfaz será en español.
RNF 2	Disponibilidad	El sistema debe estar disponible todo el tiempo para los usuarios autorizados.
RNF 3	Disponibilidad	El período entre fallos recuperables, como por ejemplo fallos en el servidor no debe exceder las 24 horas.

RNF 4	Característica del software	Sistema operativo: Nova 2011 hasta Nova 2017 Navegador web: Mozilla Firefox 2.0 Servidor web: Apache 2.2 Sistema gestor de base de datos: Elasticsearch

2.2.3 Descripción de requisitos de software

La descripción de los requisitos funcionales de la propuesta de solución se realiza mediante las historias de usuario. Este producto de trabajo permite describir las funcionalidades que el sistema debe poseer, sean requisitos funcionales o no funcionales. Cada historia de usuario es lo suficientemente comprensible y delimitada para que los programadores puedan implementarlas sin dificultad (28). A continuación, se muestra la historia de usuario del RF2 que preside de una prioridad alta, las restantes historias de usuario se mostrarán en el (anexo 3).

Tabla 3. Historia de usuario del RF2 Mostrar listado de los paquetes más populares
(Fuente: Elaboración propia)

Historia de Usuario	
Número: 1	**Nombre del requisito:** Mostrar el listado de los paquetes más populares
Programador: Dareyna de la Caridad Muñoz González	**Iteración Asignada:** 1 iteración
Prioridad: Alta	**Tiempo Estimado:** 2 semanas
Riesgo en Desarrollo: No aplica	**Tiempo Real:** 2 semanas
Descripción: La aplicación muestra un listado con los paquetes ordenados, a partir de la cantidad de veces que se han descargado. Inicialmente el usuario introduce la cantidad de paquetes que quiere mostrar. Si la cantidad especificada es mayor que la cantidad de paquetes descargados, solo se mostrarán los descargados. El usuario podrá encontrarse con tres escenarios: el primero que inserte datos vacíos por lo que la aplicación mostrará un error de código 406, el segundo escenario cuando inserte datos incorrectos mostrará el código 405 y el último escenario es el flujo normal de eventos cuando los datos son correctos.	
Observaciones. • El usuario debe autenticarse	

- Para acceder a la URL debe utilizar el método POST.
- La URL para acceder a la funcionalidad está compuesta por:

  La dirección donde esté publicado el servicio, el puerto 5000

  Ejemplo: http//:10.53.3.98:5000/app/favorite

2.2.4 Validación de requisitos de software

La validación de los requerimientos analiza la especificación a fin de garantizar que todos ellos han sido enunciados sin ambigüedades; que se detectaron y corrigieron las inconsistencias, las omisiones y los errores, y que los productos del trabajo se presentan conforme a los estándares establecidos para el producto (26). A continuación, se describen las técnicas de validación de requisitos utilizadas en la propuesta de solución:

Diseño de casos de pruebas

Los diseños de casos de pruebas permiten crear un conjunto de entradas y salidas esperadas que sean efectivos para descubrir defectos en los programas y muestren que el sistema satisface sus requerimientos. Para su diseño, se selecciona una característica del sistema o componente que se está probando, un conjunto de entradas que ejecutan dicha característica, se documentan las salidas esperadas o rasgos de salida y donde sea posible se diseña una prueba automatizada que demuestre que las salidas reales y las esperadas son las mismas (25).

En el epígrafe 3.5.1 se evidencia un ejemplo de los diseños de casos de pruebas elaborados para la validación de los requisitos funcionales de la propuesta de solución.

Revisión Técnica Formal

Una revisión técnica formal (RTF) es una actividad del control de calidad del software realizada por ingenieros de software. Los objetivos de una RTF son: descubrir los errores en funcionamiento, lógica o implementación de cualquier representación del software; verificar el cumplimiento de sus requisitos; garantizar su representación de acuerdo a los estándares predefinidos; obtener uniformidad en su desarrollo y hacer proyectos más manejables (26).

La RTF realizada a los diferentes productos de trabajo obtenidos en esta disciplina fue desarrollada por la analista y administradora de la calidad del proyecto de Nova. La misma permitió identificar las deficiencias existentes en la descripción de los requisitos de software.

2.3 Análisis y diseño

En esta disciplina se modela el sistema y su forma (incluida su arquitectura) para que soporte todos los requisitos. Los modelos desarrollados son más formales y específicos (12). En el presente epígrafe se describe el diseño de clases, datos y arquitectura de la propuesta de solución.

2.3.1 Diagrama de clases del diseño

Un diagrama de clases sirve para visualizar las relaciones entre las clases que involucran el sistema, las cuales pueden ser asociativas, de herencia y de uso (29). A continuación, se presenta el diagrama de clases del diseño de la propuesta de solución.

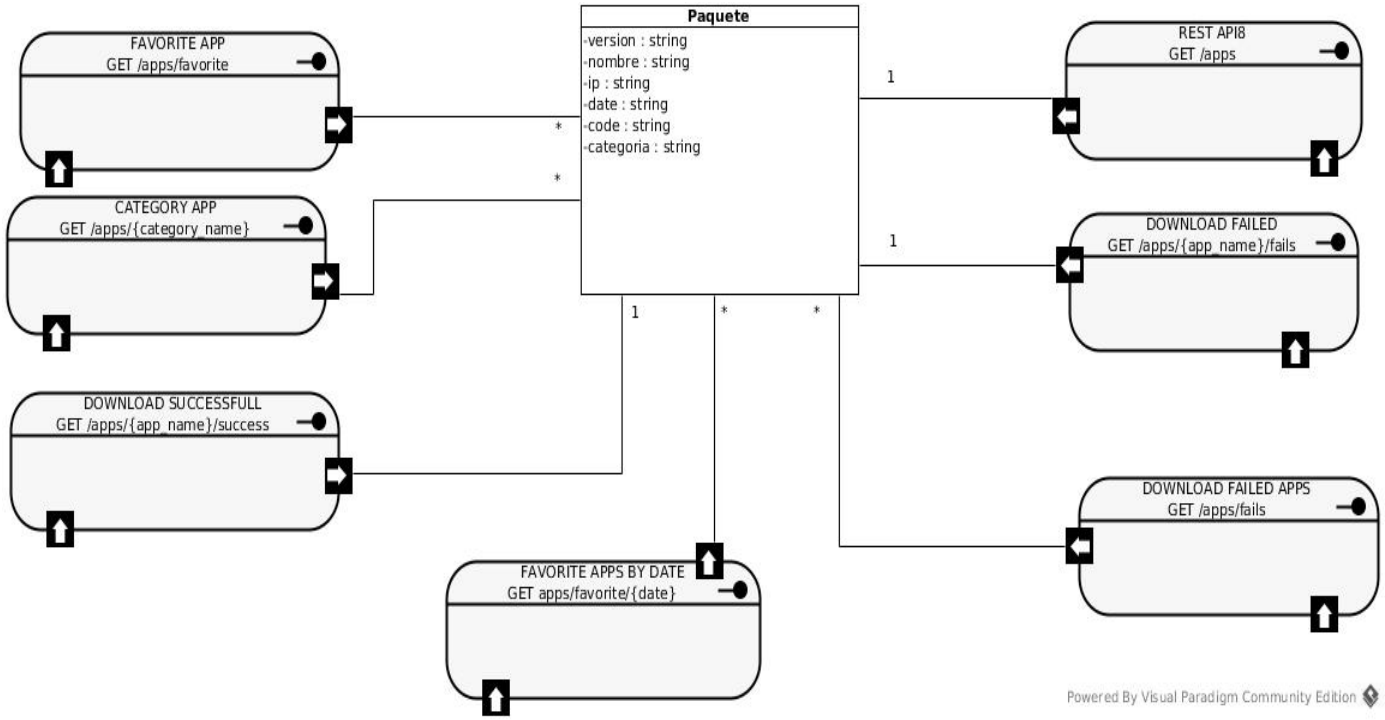


Figura 2 Diagrama de clases del diseño

Fuente: Elaboración propia)

Descripción del diagrama de clases del diseño

- Apps Favorite: es la encargada de mostrar la URL que contiene los paquetes más populares.
- Apps Category: es la encargada de mostrar la URL que contiene los paquetes por categoría.
- Apps Favorite BY DATE: es la encargada de mostrar la URL que contiene los paquetes por fecha.
- Apps Fails: es la encargada de mostrar la URL que contiene el ranking de los paquetes que fallan en la descarga.

- Apps App: es la encargada de mostrar la URL que contiene el listado de conexiones ip.
- Apps Download Success: es la encargada de mostrar la URL que contiene los paquetes que se descargan con éxito.
- Apps Download Fails: es la encargada de mostrar la URL que contiene los paquetes que fallan en la descarga.

2.3.2 Patrones de diseño

Los patrones de diseño brindan un esquema que facilita el trabajo y aportan mayor organización y claridad en la estructura de la aplicación. En el diseño de la propuesta de solución se utilizaron los patrones GRASP (Patrones de Software para la Asignación General de Responsabilidad) (26).

Se puede llegar a la conclusión que un patrón de diseño es una estructura de diseño que ayuda a estandarizar el código, haciendo que el diseño sea más comprensible para otros programadores.

Patrones GRASP

Los patrones GRASP describen los principios fundamentales de diseño de objetos para la asignación de responsabilidades. Constituyen un apoyo para la enseñanza que ayuda a entender el diseño de objeto esencial y aplica el razonamiento para el diseño de una forma sistemática, racional y explicable. El uso de estos patrones es de vital importancia para el desarrollo de una aplicación con la calidad requerida (26). A continuación, se muestran los patrones GRASP utilizados en la propuesta de solución:

Controlador: Es un evento generado por actores externos. Se asocian con operaciones del sistema, operaciones del sistema como respuestas a los eventos del sistema, tal como se relacionan los mensajes y los métodos (30). En la Figura 3 se puede observar el uso de este patrón en la propuesta de solución. Ejemplo del uso de este patrón se evidencia en las clases packageMoreFail, searchIP y moreDownloadByDate que se encargan de atender las peticiones de los usuarios.

```
def packageMoreFail(self,rank):
    return self.model.packageMoreFail(rank)

def searchIP(self):
    return self.model.searchIP()

def moreDownloaledByDate(self,rank,date):
    return self.model.moreDownloaledByDate(rank,date)
```

Controlador

Figura 3. Patrón Controlador de la propuesta de solución

(Fuente: Elaboración propia)

Alta cohesión: Asigna responsabilidades de manera tal que la cohesión siga siendo alta, o sea que las funcionalidades de las clases estén altamente relacionadas de forma tal que exista una colaboración entre ellas para compartir el esfuerzo y no caiga todo el peso sobre una única clase. Usar este patrón simplifica el mantenimiento y favorece el bajo acoplamiento (30). En la Figura 4 se evidencia el uso de este patrón en la propuesta de solución. Ejemplo del uso de este patrón se evidencia en la funcionalidad @app. route ('app/favorite/date, methods=['GET','POST']') que es la encargada de devolver la fecha dado un paquete, para ello hace uso de la funcionalidad def moreDownloaledBydate (self, rank, date), ubicado en la clase Controller.

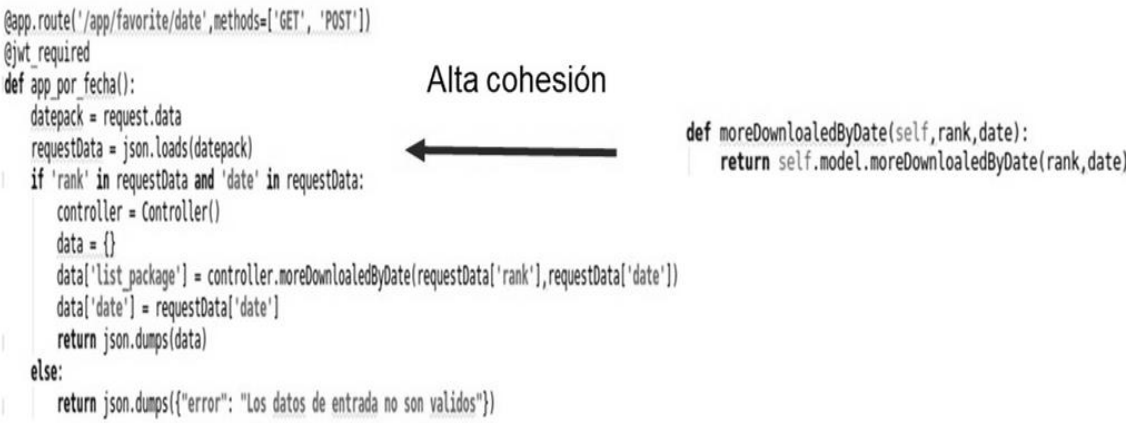


Figura 4. Patrón Alta cohesión de la propuesta de solución

(Fuente: Elaboración propia)

Experto: Es un principio básico que suele utilizarse en el diseño orientado a objetos. Con él no se pretende designar una idea oscura ni extraña; expresa simplemente la "intuición" de que los objetos hacen cosas relacionadas con la información que poseen. Ofrece una analogía con el mundo real (30). En la Figura 5 se presenta el uso de este patrón en la propuesta de solución. Ejemplo del uso de este patrón se evidencia en la clase UserModel, que es que la clase tiene acceso necesario a la información que le corresponde.

```python
class UserModel(db.Model):
    __tablename__ = 'users'
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(120), unique=True, nullable=False)
    password = db.Column(db.String(120), nullable=False)

    def save_to_db(self):
        db.session.add(self)
        db.session.commit()

    @classmethod
    def find_by_username(cls, username):
        return cls.query.filter_by(username=username).first()

    @classmethod
    def return_all(cls):
        def to_json(x):
            return {
                'username': x.username,
                'password': x.password
            }

        return {'users': list(map(lambda x: to_json(x), UserModel.query.all()))}
```

Experto

Figura 5. Patrón Experto de la propuesta de solución

(Fuente: Elaboración propia)

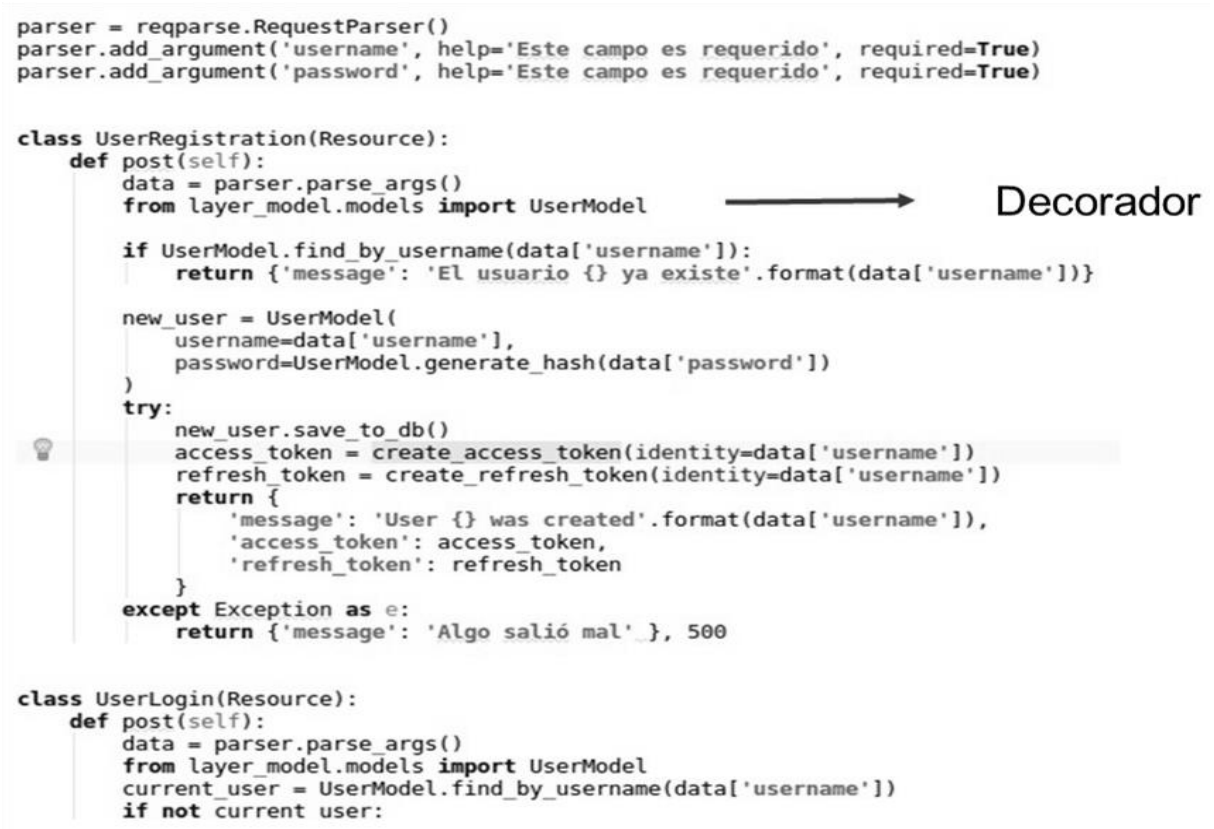
Bajo acoplamiento: El Bajo Acoplamiento es un patrón evaluativo que el diseñador aplica al juzgar sus decisiones de diseño. Estimula asignar una responsabilidad de modo que su colocación no incremente el acoplamiento tanto que produzca los resultados negativos propios de un alto acoplamiento (30). En la Figura 6 se muestra el uso de este patrón en la propuesta de solución. Ejemplo del uso de este patrón se evidencia en la clase UserLogoutRefresh, ya que se relaciona con la menor cantidad de clases posibles para el cumplimiento de su función.

Figura 6. Patrón Bajo acoplamiento de la propuesta de solución
(Fuente: Elaboración propia)

Patrones GOF

Además del uso de los patrones GRASP se tuvo en cuenta los patrones GOF, acrónimo de *Gang Of Four,* los cuales se clasifican en tres grandes categorías basadas en su propósito: creacionales, estructurales y de comportamiento (26).

Decorador: Asigna responsabilidades adicionales a un objeto dinámicamente, proporcionando una alternativa flexible a la herencia para extender la funcionalidad (31). En la Figura 7 se presenta el uso de este patrón en la propuesta de solución. Ejemplo del uso de este patrón se evidencia en la clase UseRegistration (Resource), que es que las clases tienen la función decoradora.




```
parser = reqparse.RequestParser()
parser.add_argument('username', help='Este campo es requerido', required=True)
parser.add_argument('password', help='Este campo es requerido', required=True)


class UserRegistration(Resource):
    def post(self):
        data = parser.parse_args()
        from layer_model.models import UserModel

        if UserModel.find_by_username(data['username']):
            return {'message': 'El usuario {} ya existe'.format(data['username'])}

        new_user = UserModel(
            username=data['username'],
            password=UserModel.generate_hash(data['password'])
        )
        try:
            new_user.save_to_db()
            access_token = create_access_token(identity=data['username'])
            refresh_token = create_refresh_token(identity=data['username'])
            return {
                'message': 'User {} was created'.format(data['username']),
                'access_token': access_token,
                'refresh_token': refresh_token
                }
        except Exception as e:
            return {'message': 'Algo salió mal' }, 500


class UserLogin(Resource):
    def post(self):
        data = parser.parse_args()
        from layer_model.models import UserModel
        current_user = UserModel.find_by_username(data['username'])
        if not current user:
```

Figura 7. Patrón Decorador de la propuesta de solución

(Fuente: Elaboración propia)

Singleton: Es un patrón de diseño que se utiliza para garantizar que una clase solo tenga una instancia y proporcionar un punto de acceso global a ella (2). En la aplicación es imprescindible que exista solamente una instancia de los ficheros que se van a utilizar para leer la información almacenada en el repositorio. Esta información va a ser accedida desde un solo punto, permitiendo un acceso controlado a la única instancia. Se va a generar un archivo donde se almacenan los mismos, de este archivo es necesario tener una única instancia que se va a utilizar para almacenar en la base de datos la información contenida en él. En la Figura 8 se evidencia el uso de este patrón en la propuesta de solución. Ejemplo del uso de este patrón se evidencia en la clase UserModel, que es que las clases tienen única instancia de ella que se va a utilizar para almacenar en la base de datos la información contenida en él.

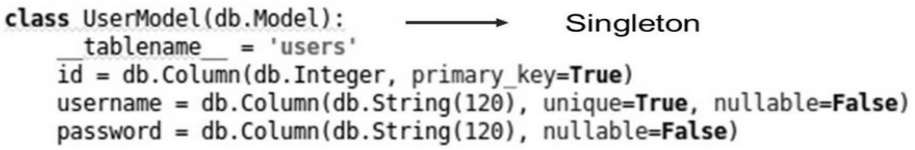


Figura 8. Patrón Singleton de la propuesta de solución

(Fuente: Elaboración propia)

2.3.3 Modelado de datos

Un modelo de datos es una descripción de una base de datos. Contiene la estructura de los datos, su tipo, descripción, la forma en que se relacionan y sus restricciones de integridad. Describe los elementos de la realidad que intervienen en un problema dado y la forma en que se relacionan esos elementos entre sí orientados a resolver un problema determinado (32).

El modelado de datos que se utilizó en la propuesta de solución, teniendo en cuenta que es una base de datos no relacional que su información se devuelve en formato JSON, se realizó siguiendo la siguiente estructura:

Estructura de datos

Log en notación JSON sería:

{

“IP”: String

“Fecha-Hora”: String

“URL que accedió”: String

“Paquete”: String

“Tiempo”: String

“Lugar donde se conectó”: String

“Versión”: String

“Método”: String

}

2.3.4 Diseño arquitectónico

El diseño arquitectónico tiene como objetivo generar propuestas e ideas para la creación y realización de espacios físicos dentro de la arquitectura de un software. Contiene patrones arquitectónicos, que se utilizan para resolver un problema de diseño de una aplicación específica dentro de un contexto determinado, sujeto a limitaciones y restricciones; y estilos arquitectónicos que permiten la organización, comunicación e integración entre los diferentes componentes de un sistema informático (26). En la propuesta de solución se utilizó el patrón MVC y el estilo REST descritos a continuación.

Patrón arquitectónico MVC

El patrón MVC (Modelo-Vista-Controlador) permite organizar y estructurar las clases en las siguientes partes: el Modelo es la capa que contiene toda la información sobre los datos; la Controladora es la capa que se encarga de la lógica de negocio, cómo acceder a los datos y validarlos teniendo en cuenta su comportamiento y relaciones, la Vista es la capa que contiene la API REST (33). A continuación se presenta la estructura del patrón MVC.

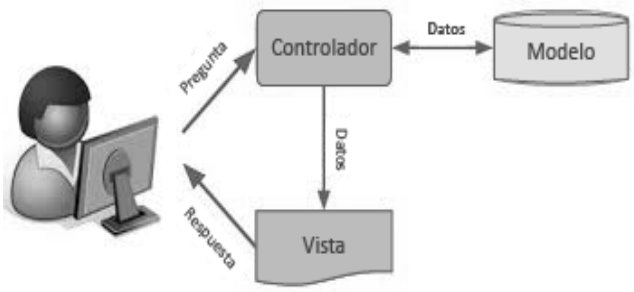


Figura 9. Patrón arquitectónico MVC

(Fuente: (34)*)*

La Figura 9 representa la arquitectura de la aplicación en el marco de trabajo según el patrón arquitectónico MVC.

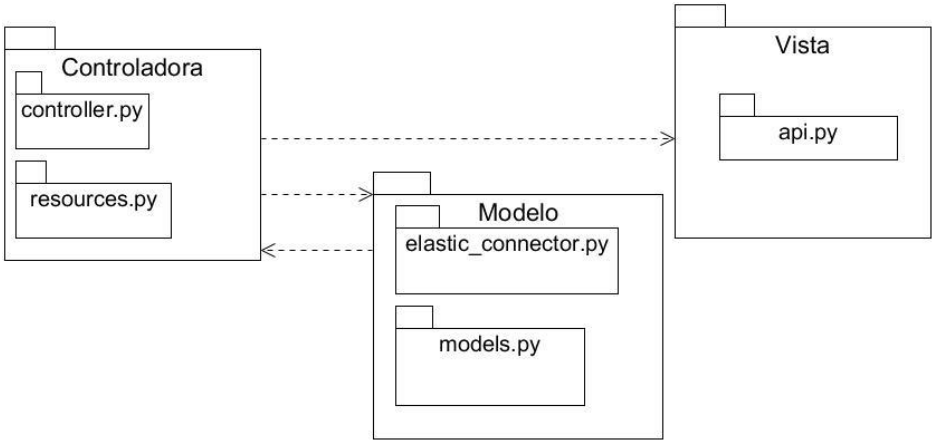


Figura 10. Diseño arquitectónico de la propuesta de solución

(Fuente: Elaboración propia)

Descripción de la arquitectura de la aplicación

Modelo: Contiene dos clases:

- La clase elastic_connector.py: es la encargada de crear las consultas DSL[4].

[4] DSL: Lenguaje de dominio específico

- La clase models.py: es la clase que se encarga de importar los paquetes.

Controlador: Contiene dos clases:

- La clase controller.py: es la encargada de proporcionar las peticiones de los usuarios, tiene los métodos que interactúan con el modelo y la vista en que se ejecutan las acciones realizadas.
- La clase resources.py: es la clase que se encarga de importar los paquetes.

Vista: Contiene una clase:

- La clase api.py: es la clase que se encarga de contener la API REST.

Estilo arquitectónico REST

El estilo arquitectónico REST permite que las comunicaciones entre productor y consumidor sean más ligeras, mantenibles y escalables. Se emplea para el desarrollo API REST empleando el protocolo HTTP, garantizando que pueda ser utilizada prácticamente por cualquier lenguaje de programación y de manera fácil. Además es un requisito de un servicio REST que el cliente y el servidor sean independientes entre sí (35). En la Figura 11 se muestra el estilo arquitectónico REST.

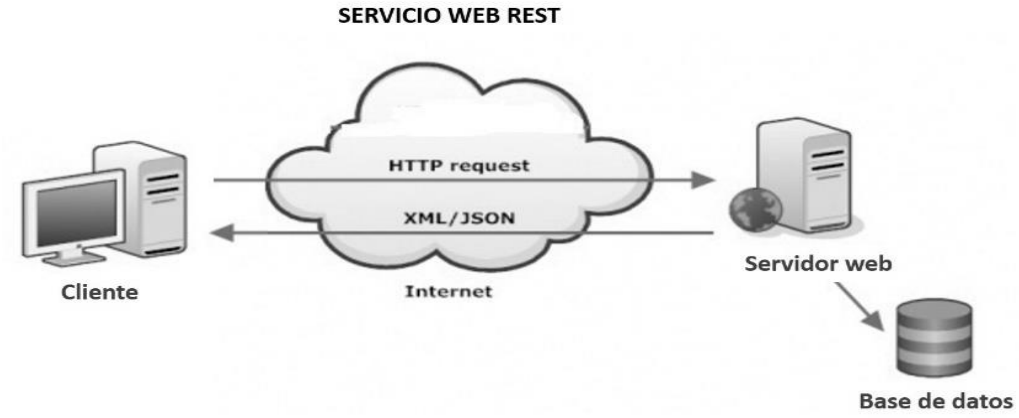


Figura 11. Estilo arquitectónico REST
(Fuente: (36)*)*

En la Figura 11 se evidencia la utilización del estilo arquitectónico REST en la propuesta de solución.

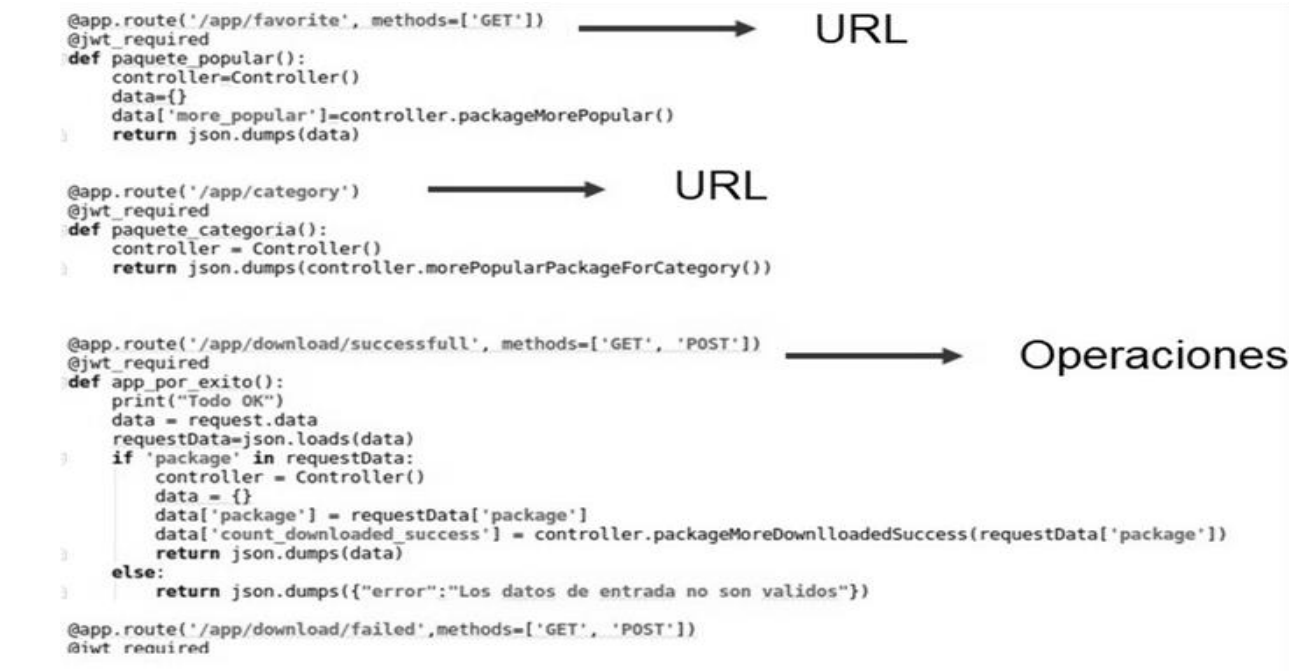


Figura 12. Estilo arquitectónico de la propuesta de solución

(Fuente: Elaboración propia)

2.5 Conclusiones parciales

A partir del análisis y diseño de la API REST se arribó a la siguiente consideración:

La elaboración del modelo conceptual permitió caracterizar el contexto del negocio a informatizar para conocer las necesidades del cliente y los usuarios finales, así como las restricciones existentes en la propuesta de solución. Se determinaron 8 requisitos funcionales y 4 no funcionales. El diseño de las clases mediante la utilización de los patrones GRASP y GOF posibilitó la obtención de un diseño que facilita el mantenimiento de la propuesta de solución. El diseño arquitectónico basado en el patrón MVC y el estilo REST facilitó la comunicación entre los diferentes componentes de la API REST implementada.

CAPÍTULO 3: Implementación y evaluación de la API asociada al uso del repositorio de Nova

Introducción

La etapa de implementación constituye una parte fundamental durante el proceso de desarrollo de un software, es en este momento donde se define y se organiza el código de la propuesta de solución. Durante esta etapa se materializan en forma de código las descripciones y arquitectura propuestas en la etapa de análisis y diseño; conformando así el producto final requerido por el cliente.

Teniendo en cuenta que todo software debe ser puesto a prueba, garantizándose que se cumplan con todos los estándares requeridos y con todas las condiciones y funcionalidades que el cliente necesita. A esta etapa se le conoce como validación del sistema y en ella, pueden realizarse diferentes tipos de pruebas en función de los objetivos de las mismas

3.1 Estándares de codificación utilizados

Se definen estándares de codificación porque un estilo de programación homogéneo en un proyecto permite que todos los participantes lo puedan entender en menos tiempo y que el código en consecuencia sea mantenible. La codificación de las aplicaciones fue realizada en inglés y siguiendo los estándares que se explican a continuación (37).

Indentación: Emplear 4 espacios por cada nivel de indentación.

Tabuladores y espacios: No mezclar las tabulaciones con los espacios

```
for hit in query:
    list.append(hit['path'])

listPack = []
for pack in list:
    listPack.append(self.getPackage(pack))

packMorePopular=None
countOfMoreFound=0
for currentPack in listPack:
    count= 0
    for findPack in listPack:
        if currentPack==findPack:
            count=count+1

    if count>countOfMoreFound:
        countOfMoreFound=count
        packMorePopular=currentPack

return packMorePopular
```

Figura 13. Tabuladores y espacios de la propuesta de solución

(Fuente: Elaboración propia)

Tamaño máximo de líneas: Se limitan todas las líneas a un máximo de 50 caracteres. Esto puede ser realizado mediante el uso de paréntesis de forma implícita o mediante el uso de la barra invertida.

Nomenclatura de las funciones: Los nombres de funciones deberán estar escritos en letras minúsculas, en el caso de que sea una palabra compuesta empezará con mayúscula en la segunda palabra. Ejemplo:

```
def packageMoreDownlloadedFail(self,package):
    return self.model.packageMoreDownlloadedFail(package)

def packageMoreFail(self,rank):
    return self.model.packageMoreFail(rank)

def searchIP(self):
    return self.model.searchIP()

def moreDownloaledByDate(self,rank,date):
    return self.model.moreDownloaledByDate(rank,date)
```

Figura 14. Nomenclatura de la propuesta de solución

(Fuente: Elaboración propia)

3.2 Diagrama de despliegue

El diagrama de despliegue permite modelar la disposición física de un sistema. Muestra el hardware usado y los componentes instalados en el hardware, además de las conexiones físicas entre este y las relaciones entre componentes. Es utilizado para capturar los elementos de configuración del procesamiento y las conexiones entre esos elementos. También se utiliza para visualizar la distribución de los componentes de software en los nodos físicos (38). En la Figura 15 que se muestra a continuación se representa el diagrama de despliegue de la propuesta de solución.

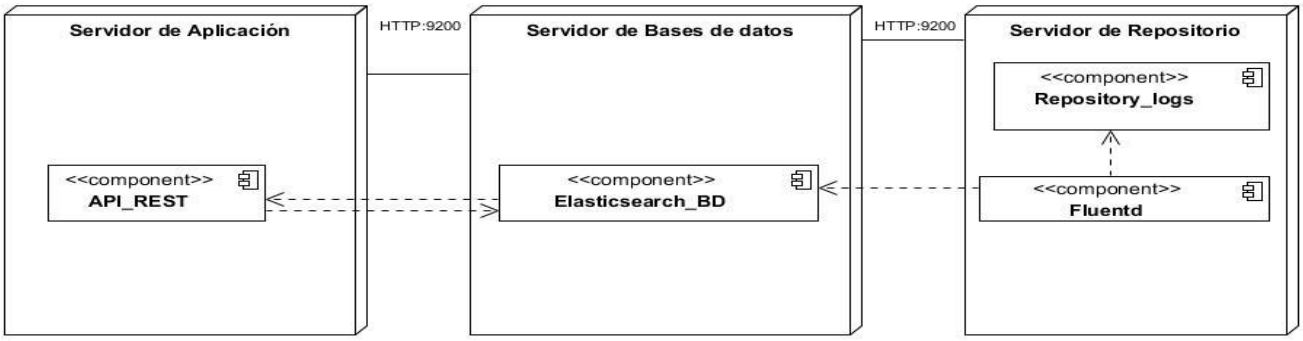


Figura 15. Diagrama de despliegue de la propuesta de solución

(Fuente: Elaboración propia)

Descripción de los nodos del diagrama de despliegue

- Servidor de Aplicación: contendrá la API REST implementada.
- Servidor de Bases de datos: proporciona los servicios de la base de datos no relacional Elasticsearch donde están alojados todos los datos.
- Servidor de Repositorio: Repositorio de Nova Servidores y que proporciona los paquetes necesarios en el modelo. Además, contiene la herramienta que permite conectarse a la base de datos insertando los logs del repositorio en tiempo real conocida como *Fluentd.*

3.3 Pruebas de software para la evaluación de la propuesta de solución

Las pruebas de desarrollo de software son un conjunto de herramientas, técnicas y métodos que evalúan la excelencia y el desempeño de un software, involucran las operaciones del sistema bajo condiciones controladas y evaluando los resultados. Las técnicas para encontrar problemas en un programa son variadas y van desde el uso del ingenio por parte del personal de prueba hasta herramientas automatizadas que ayudan a aliviar el peso y el costo de tiempo de esta actividad (26).

La metodología de desarrollo de software AUP para la UCI establece 3 disciplinas para la ejecución de pruebas: internas, liberación y aceptación. Como parte de las pruebas se decide la realización de las pruebas internas y de aceptación, se descartan las pruebas de liberación ya que estas son diseñadas y ejecutadas por una entidad certificadora de la calidad externa. En las pruebas internas se propone verificar el resultado de la aplicación. A continuación, se describen los tipos de pruebas de software aplicadas, así como los métodos y técnicas empleadas para la evaluación de la propuesta de solución. En el caso de las pruebas de aceptación, se llevan a cabo para verificar que el software está listo y que puede ser utilizado por usuarios finales, para ejecutar las tareas y funciones para las que fue construido.

3.4 Aplicación de las pruebas de software

En el epígrafe se describen las pruebas de software realizadas en las disciplinas de pruebas internas y pruebas de aceptación propuestas en la metodología de desarrollo de software AUP para la UCI.

3.5. Pruebas internas

En esta disciplina se verifica el resultado de la implementación probando cada construcción, incluyendo tanto las construcciones internas como intermedias, así como las versiones finales a ser liberadas (12). En la disciplina se aplicaron pruebas unitarias mediante el método de caja blanca y la técnica de camino básico utilizando el producto de trabajo diseño de casos de pruebas. A continuación, se describen los pasos de la aplicación de esta prueba.

3.5.1 Pruebas unitarias

Las pruebas unitarias se centran en probar cada componente de código de un software de forma individual para asegurar que funcione de manera apropiada como unidad. Emplean técnicas de prueba que recorren caminos específicos en la estructura de control de los componentes (pruebas estructurales) (26). El método de prueba utilizado para la realización de esta prueba es Caja blanca y la técnica de prueba contenida en este método que se empleó fue la Técnica del camino básico.

Método de prueba: Caja blanca

El método de Caja blanca se enfoca en probar el sistema teniendo en cuenta la estructura interna del mismo. Verifica la correcta implementación de las unidades internas, las estructuras y sus relaciones y hacen énfasis en la reducción de errores internos (26) .

Técnica de prueba: Camino básico

La técnica del Camino básico permite obtener una medida de la complejidad lógica de la codificación de software y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución independiente en un componente o programa. Un camino o ruta es una vía por la cual procede la ejecución a través de una función desde su inicio hasta el fin (26) .

Pasos para la aplicación de la técnica del camino básico:

1. Dibujar el grafo de flujo de la funcionalidad o procedimiento a analizar

```
def packageMoreDownlloadedSuccess(self,package):
    query = Search(using=self.db)
    query = Search().using(self.db).query("match", path=package)
1   query.aggs.bucket('per_tag', 'terms', field='tags') \
        .metric('max_lines', 'max', field='lines')
    list = []
2   for hit in query:
3       list.append(hit['code'])
4
5   count=0
6   for data in list:
7       if data == 200:
8           count = count + 1
9
10
11  return count
```

Figura 16. Código de la implementación del RF 3. Mostrar la cantidad de veces que se descarga una aplicación con éxito

(Fuente: Elaboración propia)

La Figura 17 presenta el grafo de flujo obtenido del código representado en la Figura 16.

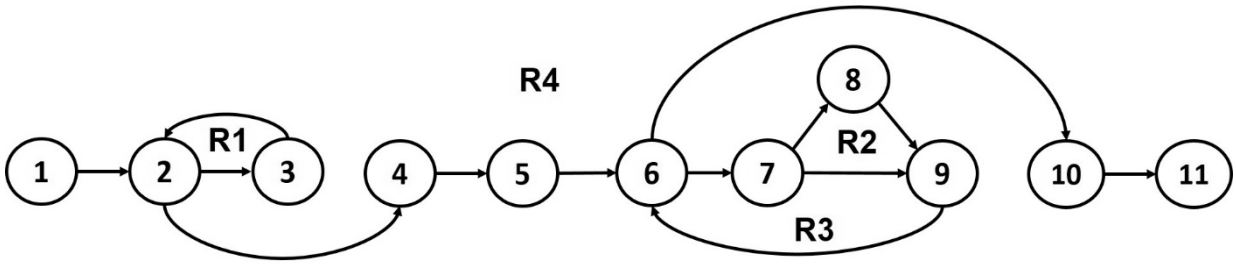


Figura 17. Grafo de flujo del código de la implementación del RF 3. Mostrar la cantidad de veces que se descarga una aplicación con éxito

(Fuente: Elaboración propia)

2. Determinar la complejidad ciclomática del grafo de flujo resultante

La complejidad ciclomática del grafo (V(G)) se puede calcular de las tres formas siguientes:

- V(G) = A - N + 2
  V(G) = 13 – 11 + 2 = 4
  A: número de aristas del grafo de flujo
  N: número de nodos del grafo

- V(G) = P + 1
  V(G) = 3 + 1 = 4
  P: número de nodos predicados (nodos con más de una arista de salida) contenidos en el grafo

- V(G) = Regiones (R)
  = 4
  R: son las áreas delimitadas por nodos y aristas en el grafo

3. Determinar el conjunto básico de caminos linealmente independientes

- Camino básico 1: 1,2,4,5,6,7,8,9,6,10,11
- Camino básico 2: 1,2,4,5,6,7,9,6,10,11
- Camino básico 3: 1,2,4,5,6,10,11
- Camino básico 4: 1,2,3,2,4,5,6,7,8,9,6,10,11

4. Definir los casos de prueba para comprobar la ejecución de cada camino del conjunto básico

En el diseño de los casos de prueba se especifican los siguientes elementos:

- Descripción: contiene una descripción sobre las restricciones de los datos de entrada que debe tener el caso de prueba.
- Condición de ejecución: se especifican los parámetros que debe poseer el caso de prueba para que se cumpla una condición deseada como respuesta del funcionamiento del procedimiento.
- Entrada: se muestran los parámetros de entrada al procedimiento.

- Resultados esperados: se explica el resultado esperado de la ejecución del procedimiento.

La Tabla 4 muestra el diseño de casos de pruebas para el camino 4 correspondiente al requisito funcional Mostrar la cantidad de veces que se descarga una aplicación con éxito. En este camino se prueba que se muestren de forma satisfactoria las descargas exitosas del repositorio.

Tabla 4. Diseño de casos de pruebas para el camino básico 4 del RF 3. Mostrar la cantidad de veces que se descarga una aplicación con éxito

(Fuente: Elaboración propia)

Diseño de caso de prueba para el camino 4	
Descripción	Muestra la cantidad de veces que se descarga una aplicación con éxito
Condición de ejecución	Se introduce el nombre del paquete que se desea mostrar las descargas exitosas que ha tenido
Entrada	Nombre del paquete: php7.0
Resultados esperados	Mostrar la cantidad de veces que fue descargado exitosamente el paquete

A partir de la aplicación de los casos de pruebas se comprobó que el flujo de trabajo de las funcionalidades es correcto, ya que cada sentencia fue ejecutada al menos una vez, cumpliéndose así las condiciones de las pruebas y los resultados esperados.

3.5.2 Pruebas funcionales

Las pruebas funcionales tienen como objetivo ejercitar profundamente el sistema comprobando la integración del sistema de información globalmente, verificando el funcionamiento correcto de las interfaces entre los distintos subsistemas que lo componen y con el resto de sistemas de información con los que se comunica. Se seleccionó este tipo de prueba con el objetivo de verificar y valorar las funcionalidades del sistema mediante la especificación de requisitos (26). El método de prueba utilizado para la realización de esta prueba es Caja negra y la técnica de prueba contenida en este método que se empleó fue la de Partición de equivalencia.

Método de prueba: Caja negra

Se enfoca en probar el sistema sin tomar en cuenta la estructura interna del mismo, su objetivo es validar que las salidas sean las esperadas. Se centra en encontrar las circunstancias en las que el sistema no se comporta conforme a las especificaciones establecidas.

Técnica de prueba: Partición de equivalencia

La técnica divide el dominio de entrada de un programa en clases de datos, a partir de las cuales pueden derivarse casos de prueba. Además, descubre clases de errores, que, de otra manera, requeriría la ejecución de muchos casos antes de que se observe el error general. Mediante su empleo se puede reducir al máximo el total de casos de prueba que deben desarrollarse (37).

Diseños de caso de pruebas

Es una parte de las pruebas de componentes y sistemas en las se diseñan los casos de prueba (entradas y salidas esperadas) para probar el sistema. Su objetivo es crear un conjunto de casos de prueba que sean efectivos descubriendo defectos en los programas y muestren que el sistema satisface sus requerimientos. Para su diseño, se selecciona una característica del sistema o componente que se está probando, un conjunto de entradas que ejecutan dicha característica, se documentan las salidas esperadas o rasgos de salida y donde sea posible se diseña una prueba automatizada que demuestre que las salidas reales y las esperadas son las mismas (25). A continuación, se presenta el diseño de casos de prueba para el requisito funcional 2. Mostrar el paquete más popular.

Abreviaturas utilizadas:
CP: Caso de Prueba.
HU: Historia de Usuario.
EC: Escenario.

Tabla 5 Caso de Prueba Funcional del RF2 Mostrar listado de los paquetes más populares

(Fuente: Elaboración propia)

Caso de Prueba Funcional	
CP1_HU2	**HU_2: Mostrar el paquete más popular**
Responsable: Dareyna Muñoz González	
Descripción: El caso de prueba se inicia luego que el usuario entre a la aplicación. En la sección donde se inicia la URL, el usuario debe escoger el endpoint que define la URL que desea ver, ya una vez escogida la URL, en este caso el usuario introduce la cantidad de paquetes que quiere mostrar.	

Escenario	Descripción	Respuesta del sistema	Flujo del sistema
EC 1.1 Mostrar el paquete más popular.	El usuario selecciona la URL una vez seleccionada, introduce la cantidad de paquete a mostrar.	Se muestra la información de los paquetes	El usuario introduce la cantidad de paquete a mostrar.
EC 1.2 Error cuando introduce datos incorrectos.	El usuario selecciona la URL una vez seleccionada, introduce datos incorrectos.	Se mostrará un error de código 405	El usuario introduce la cantidad de paquete a mostrar
EC 1.2 Error cuando introduce datos vacíos	El usuario selecciona la URL una vez seleccionada, introduce datos vacíos.	Se mostrará un error de código 406.	El usuario introduce la cantidad de paquete a mostrar

3.6 Pruebas de aceptación

Es la prueba final antes del despliegue del sistema. Su objetivo es verificar que el software está listo y que puede ser usado por usuarios finales para ejecutar aquellas funciones y tareas para las cuales el software fue construido (12). En la presente investigación se decidió realizar la prueba de aceptación ya que es necesario para la evaluación de la solución que el cliente esté de acuerdo con el funcionamiento de la API REST.

La evaluación realizada por el cliente consistió en la ejecución de pruebas funcionales. Las recomendaciones realizadas fueron solucionadas y para constatar la conformidad del cliente hacia la aplicación desarrollada emitió un acta de aceptación de productos de software, la misma puede ser consultada en el (anexo 6).

3.7 Técnica de Iadov

La técnica de Iadov es utilizada para determinar el nivel de satisfacción individual y grupal de los usuarios a partir de una encuesta elaborada. La aplicación de esta técnica es una vía para el estudio de satisfacción, los criterios que se utilizan se fundamentan en las relaciones que se establecen entre las tres preguntas cerradas, que se intercalan dentro de un cuestionario y cuya relación el encuestado desconoce (39).

La encuesta elaborada para evaluar el índice de satisfacción de los usuarios potenciales de la propuesta de solución fue aplicada a 7 especialistas del proyecto Nova. Las preguntas 2,3 y 4 de la encuesta se

relacionan a través de lo que se denomina el “Cuadro Lógico de Iadov” que se muestra en la siguiente tabla.

Tabla 6. Cuadro Lógico de Iadov

(Fuente: Elaboración propia)

4. Luego de haber mostrado los resultados de la solución refleje en qué medida le gusta la solución desarrollada.	2. ¿Considera usted correcta la forma en que se analizan los logs generados por el repositorio de Nova actualmente?								
	No			No sé			Sí		
	3. ¿Considera usted factible la implementación de una API REST que permita conocer el estado de los paquetes del repositorio de Nova?								
	Sí	No sé	No	Sí	No sé	No	Sí	No sé	No
Me gusta mucho	1	2	6	2	6	6	6	6	6
Me gusta más de lo que me disgusta	2	2	3	2	3	3	6	3	3
Me da lo mismo	3	3	3	3	3	3	3	3	3
Me disgusta más de lo que me gusta	6	3	6	3	4	4	3	3	4
No me gusta nada	6	6	6	6	4	4	6	4	5
No sé decir	2	3	6	3	3	3	6	3	4

La forma de utilizar la tabla es la siguiente:

Cada encuestado recibe una evaluación individual en dependencia de las respuestas que dé a las preguntas cerradas. Para obtener el índice de satisfacción grupal (ISG) se trabaja con los diferentes niveles de satisfacción que se expresan en la escala numérica que oscila entre +1 (máxima satisfacción) y -1 (máxima insatisfacción). El número resultante de la interrelación de las tres preguntas indica la posición de cada encuestado en la siguiente escala de satisfacción: clara satisfacción +1, más satisfecho que insatisfecho 0.5, no definido y contradictorio 0, más insatisfecho que satisfecho -0.5 y clara insatisfacción -1. El ISG se calcula mediante la siguiente fórmula:

$$ISG = \frac{\mathrm{A}(+1) + \mathrm{B}(+0.5) + \mathrm{C}(0) + \mathrm{D}(-0.5) + \mathrm{E}(-1)}{N}$$

En esta fórmula A, B, C, D, E, representan la cantidad de encuestados colocados respectivamente en las posiciones de satisfacción 1; 2; 3 o 6; 4; 5 y donde N representa la cantidad total de encuestados.

Resultados obtenidos

1. Los resultados obtenidos de la aplicación de la encuesta se presentan en la Tabla 6.

Tabla 7. Resultados obtenidos de los encuestados

(Fuente: Elaboración propia)

Categorías grupales de satisfacción	N=7	Escala
Clara satisfacción	4	A
Más satisfecho que insatisfecho	2	B
No definido	0	C
Más insatisfecho que satisfecho	0	D
Clara de insatisfacción	0	E
Contradictorio	1	C

2. Cálculo del Índice de Satisfacción Grupal

ISG=A (+1) +B (+0.5) + C (0) /N

ISG= (4(+1) +2(+0.5)) +1(0) /7=0.71

3. Interpretación del resultado del ISG

El valor obtenido del ISG fue 0.71 lo que indica máxima satisfacción de los usuarios con respecto a la API REST para los servicios web asociados al repositorio de Nova. Se puede afirmar que se cumplió el objetivo general de la investigación. Las respuestas a las preguntas abiertas brindadas por los encuestados reafirman los beneficios que traerá la utilización del sistema informático propuesto.

Conclusiones parciales

La realización de los diferentes productos de trabajo relacionados con la implementación y evaluación de la propuesta de solución permitieron obtener un software que responden a las necesidades del cliente y se puede concluir lo siguiente:

En la informatización del API REST asociada al repositorio de Nova para la obtención de la información sobre el uso de este repositorio se tuvo en cuenta 8 requisitos funcionales y 4 no funcionales, así como las restricciones de diseño y los estándares de codificación. Se aplicaron pruebas unitarias para comprobar el flujo de trabajo de las funcionalidades implementadas, demostrando que cada camino se ejecutó de forma satisfactoria. Las pruebas de aceptación realizadas le permitieron al cliente evaluar en

el API REST el cumplimiento de los requisitos funcionales. Se evidenció la alta satisfacción del cliente hacia la solución desarrollada con un ISG de 0.71 mediante la aplicación de la técnica de Iadov.

CONCLUSIONES GENERALES

El estudio realizado sobre el uso de los repositorios de sistemas operativos GNU/Linux y los servicios web en conjunto con la entrevista realizada a los desarrolladores, posibilitó determinar las funcionalidades necesarias para la propuesta de solución.

El análisis y diseño de la propuesta de solución permitió definir la estructura que iba a contener el sistema mediante la realización de los diagramas de clase del diseño y la utilización de los patrones arquitectónicos.

La implementación de la propuesta de solución permitió el desarrollo de una API REST que brinda información acerca del repositorio de la distribución cubana de GNU/Linux Nova, conteniendo funcionalidades principales tales como: listar paquetes por categoría, listar paquetes más populares, listar aplicaciones más descargadas por categoría.

Las pruebas de software realizadas permitieron evaluar el funcionamiento de la API REST, la conformidad con los requisitos definidos y la eliminación de las no conformidades encontradas en las iteraciones, validando de esta forma el correcto funcionamiento de la propuesta de solución. La técnica de Iadov arrojó la máxima satisfacción de los usuarios con respecto al API REST para los servicios web asociados al repositorio de Nova.

RECOMENDACIONES

Se recomienda incluir la API REST en el Centro de Software de Nova.

Referencias

1. **Baig, Roger; Auli, Francesc.** *Software Libre.* s.l. : Fundació per a la Universitat Oberta de Catalunya, 2003.

2. **Peiso, Manuel Enrique.** Identificación de requisitos a partir de un repositorio de aplicaciones.Tesis para optar por el título de Ingeniero en Ciencias Informática, Universidad de las Ciencias Informáticas. [En línea] 2015. https://repositorio.uci.cu/jspui/bitstream/123456789/7061/1/TD_07878_15.pdf.

3. **Fernandez, Yusleydi.** Metodología para desarrollar la distribución cubana de GNU/Linux Nova. Centro de Software Libre y Código Abierto, Universidad de las Ciencias . [En línea] https://repositorio_institucional.uci.cu/jspui/bitstream/ident/8096/1/TM_07188_14.pdf.

4. **Salomón, Rafael R.** *El gran libro de Debian Gnu/Linux.* 2012.

5. **Todos los derechos reservados. Desarrollado por la UCI.** Nova. [En línea] 2017. https://www.nova.cu/es.

6. **Senso, Jose y Piñero, Antonio.** *El concepto de metadato. Algo más que descripción de recursos electrónicos.* s.l. : España, 2003.

7. **Open Source.** Gestores de Paquetes. [En línea] 2015. https://es.opensuse.org/Gesti%C3%B3n_de_paquetes.

8. **Dept. Ciencia de la Computación e IA All rights reserved.** Introducción a los Servicios Web. Invocación de servicios web SOAP. [En línea] 2014. www.jtech.ua.es/j2ee/publico/servc-web-2012-13/sesion01-apuntes.html.

9. **Adrerrs.** Hipertextual. [En línea] 2016. https://hipertextual.com/archivo/2014/05/que-es-api/.

10. **BBVAOpen4U.** Api Rest que es y cuales son sus ventajas. [En línea] 2016. https://bbvaopen4u.com/es/actualidad/api-rest-que-es-y-cuales-son-sus-ventajas-en-el-desarrollo-de-proyectos.

11. **Marques, Asier.** Conceptos de Api Rest. [En línea] 2015. http://asiermarques.com/2013/conceptos-sobre-apis-rest/.

12. **Rodriguez Sanchez, T.** Metodología de desarrollo para la Actividad productiva de la UCI.Universidad de las Ciencias Informáticas. La Habana. Cuba . [En línea] 2015. https://repositorio_institucional.uci.cu/jspui/handle/ident/8917.

13. **Hernandez Barrera, Yerandy.** Enriquecimiento de una Ontología Geográfica, por medios semiautomáticos, a partir de fuentes de información disponibles en Internet.Trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas. [En línea] 2016. https://repositorio_institucional.uci.cu/jspui/handle/123456789/7408.

14. **Rouse, Margaret.** SearchDataCenter en Español. [En línea] 2014. http://searchdatacenter.techtarget.com/es/definicion/Framework.

15. **Github.** Flask. [En línea] https://github.com/pallets/flask.

16. **Softwaredoit.** Qué es un Lenguaje de Programación. [En línea] https://www.softwaredoit.es/definicion/definicion-lenguaje-de-programacion.h.

17. **Escarret, Daniel Rolando.** Herramienta de análisis del repositorio de Nova.Trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas. [En línea] 2016.

18. **Fernandez, Pablo.** *Creación de una API y una interfaz web para generar un DASHBOARD A PARTIR DE LA HERRAMIENTA PERCEVAL.* 2016.

19. **Verras, Dairon.** Framework de seguridad para Nova Servidores".Trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas. [En línea] 2016. https://repositorio_institucional.uci.cu/jspui/handle/123456789/7555.

20. **JetBrains** s.r.o. Developed with drive and IntelliJ IDEA. [En línea] https://www.jetbrains.com/pycharm/.

21. Huerta, Francisco. Introduccion a Elasticsearch. [En línea] 2015. http://www.davinci-ti.es/introduccion-a-elasticsearch-y-como-instalarlo/.

22. Altarade, Mohamad. Base de dato NoSQL. [En línea] https://www.toptal.com/database/the-definitive-guide-to-nosql-databases.

23. Primeros paso con Elasticearch. [En línea] 2015. https://www.busindre.com/primeros_pasos_con_elasticsearch.

24. Palacio, Luis. Fluentd. [En línea] 2015. https://github.com/LuisPalacios/base-fluentd.

25. Sommerville, Ian. Ingeneria del Software Novena ed. España . España : s.n., 2011. ISBN:9786073206044.

26. Pressman, Roger. Ingeniería del software. New York : EUA, : s.n., 2010. ISBN:978-0-07-337597..

27. Guerra, Cesar Arturo. Obtención de Requerimientos. Técnicas y Estrategia. [En línea] 2017. https://sg.com.mx/revista/17/obtencion-requerimientos-tecnicas-y-estrategia.

28. Carlo, Letelier y Jose. Metodologías Ágiles en el Desarrollo de Software. [En línea] ., 2011.

29. Tutorial de UML. [En línea] https://users.dcc.uchile.cl/~psalinas/uml/modelo.html.

30. Craig, Larman. UML Y PATRONES. Introducción al análisis y diseño orientado a objetos. s.l. : PRENTICE HAL. 1999.

31. Belmonte Fernàndez, Òscar. Programaciòn Avanzada Patrones de Diseño. [En línea] 2015. http://www3.uji.es/~belfern/Docencia/Presentaciones/ProgramacionAvanzada/Tema2/decorador.html#36.

32. Martin's , Jame. Information Engineering , Prentice Hall,. [En línea] 1990. https://sites.google.com/site/jalexiscv/modelosdedatos.

33. Cabrera Martinez, Aimet. Modulo de procesamiento estadistico para el apoyo a la toma de decisiones del motor de busqueda Orion.Trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas. [En línea] 2017. https://repositorio_institucional.uci.cu/jspui/bitstream/123456789/7657/1/09-Aimet-Procesamiento%20estadistico.pdf.

34. Valdeón, Fernando. Programo. [En línea] 2018. https://www.programoergosum.com/cursos-online/paginas-web/138-introduccion-al-framework-de-codeigniter/introduccion-codeigniter.

35.Cano, Christian. La Arquitectura Rest. [En línea] 2017. http://www.tsgroup.com.co/wps/portal/tsg/blog/detalle-blog/la-arquitectura-rest.

36. Parvez. Types of Web Services SOAP,XML-RPC and Restful. [En línea] 2017. https://www.phpflow.com/php/web-service-types-soapxml-rpcrestful/.

37. Montano Martínez , Ronny. Herramienta de sincronización entre los sistemas operativos GNU/Linux Nova y Android.Trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas. [En línea] 2016. https://repositorio_institucional.uci.cu/jspui/bitstream/123456789/7531/1/TD_08439_16.pdf.

38. Acosta, Tòmas y Hernàndez, Nodelvis. Módulo de reportes webmétricos para el motor de búsqueda Orión. Trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas. . [En línea] junio de 2013. https://repositorio_institucional.uci.cu/ jspui/handle/ident/8296.

39. Santana, Aylin y Almanza, Erichle. Capa de Servicios Web para la aplicación FCM-Decision. [En línea] 2014. https://repositorio_institucional.uci.cu/jspui/handle/ident/9191.

40.-Representational+State+Transfe. [En línea] http://bibing.us.es/proyectos/abreproy/11247/fichero/Memoria%252F8-Representational+State+Transfer+%28REST%29.pdf.

41.Busindre. Primeros pasos con Elasticsearch. **[En línea] https://www.busindre.com/primeros_pasos_con_elasticsearch.**

42. Cáceres, Edmundo A. Analisi y diseño en el sistema de informacion . 2014.

43. Plaza, Sheila, Ramìrez, Nerea y Acosta, Carmen. API de servicios web orientados a accesibilidad. [En línea] 2015-2016. prints.ucm.es/38686/1/Memoria_API de servicios web de accesibilidad.pdf.

43. Valdeón, Fernando. ***Programo.*** [En línea] 2018. https://www.programoergosum.com/cursos-online/paginas-web/138-introduccion-al-framework-de-codeigniter/introduccion-codeigniter.

ANEXOS

Anexo 1: Entrevista realizada a especialistas del proyecto Nova de CESOL

Objetivo: obtener información sobre el uso del repositorio de la distribución cubana GNU/Linux Nova, así como requisitos funcionales y no funcionales de la propuesta de solución.

1. ¿Cree usted que son importantes los servicios web asociados al repositorio de Nova?
2. ¿Cuáles son los aspectos fundamentales a tener en cuenta cuando se consume un servicio web?
3. ¿Cree usted necesario la creación de una API REST que permita realizar estos servicios? ¿Por qué?
4. ¿Cuáles son las características que tendría este servicio?

Anexo 2: Guía de observación al proceso de mantenimiento y control del repositorio de Nova

Observador: Dareyna de la Caridad Muñoz González

Lugar: Laboratorio del proyecto Nova

Objetivo: conocer la estructura del repositorio de Nova, la agrupación de los paquetes, el ciclo de vida de un paquete, cómo se realizan las descargas de paquetes y la gestión de los logs del repositorio.

1. ¿Qué información brinda actualmente el repositorio acerca de sus aplicaciones?

2. Existe alguna forma de conocer cuándo una aplicación tiene una descarga fallida.

3. ¿Qué información contienen los log de Apache?

Anexo 3: Historias de Usuario

Mostrar el listado de paquetes

Tabla 8 Historia de usuario del RF1 Mostrar el listado de paquetes

(Fuente: Elaboración propia)

Historia de Usuario	
Número: 1	**Nombre del requisito: Mostrar el listado de paquetes**
Programador: Dareyna de la Caridad Muñoz González	**Iteración Asignada:** 2 iteración
Prioridad: Alta	**Tiempo Estimado:** 1 semana
Riesgo en Desarrollo: N/A	**Tiempo Real:** 2 semanas
Descripción: La aplicación muestra un listado con los paquetes descargados existentes en la base de datos y la categoría asociada a cada uno de ellos, en caso de que tenga una categoría asociada.	
Observaciones. • El usuario debe autenticarse. • Para acceder a la URL debe utilizar el método GET • La URL para acceder a la funcionalidad está compuesta por: La dirección donde esté publicado el servicio, el puerto 5000 Ejemplo: http//:10.53.3.98:5000/app/category	

Mostrar la cantidad de veces que se descarga un paquete con éxito

Tabla 9 Historia de usuario del R3 Mostrar la cantidad de veces que se descarga un paquete con éxito

(Fuente: Elaboración propia)

Historia de Usuario	
Número: 1	**Nombre del requisito: Mostrar la cantidad de veces que se descarga un paquete con éxito**

Programador: Dareyna de la Caridad Muñoz González	**Iteración Asignada:** 1 iteración
Prioridad: Alta	**Tiempo Estimado:** 1 semana
Riesgo en Desarrollo: N/A	**Tiempo Real:** 1 semana
Descripción: La aplicación dado un paquete muestra la cantidad de descargas exitosas que ha tenido el mismo desde los repositorios. El usuario podrá encontrarse con tres escenarios: el primero que inserte datos vacíos por lo que la aplicación mostrará un error de código 406, el segundo escenario cuando inserte datos incorrectos mostrará el código 405 y el último escenario es cuando ocurre el flujo normal de eventos cuando los datos son correcto.	
Observaciones. • El usuario debe autenticarse. • Para acceder a la URL debe utilizar el método POST. • La URL para acceder a la funcionalidad está compuesta por: La dirección donde esté publicado el servicio, el puerto 5000 Ejemplo: http//:10.53.3.98:5000/app/download/successfull	

Mostrar listado de la cantidad de veces que un paquete tiene descarga fallida

Tabla 10 Historia de usuario del RF4 Mostrar listado de la cantidad de veces que un paquete tiene descarga fallida (Fuente: Elaboración propia)

Historia de Usuario	
Número: 1	**Nombre del requisito: Mostrar listado de la cantidad de veces que un paquete tiene descarga fallida**
Programador: Dareyna de la Caridad Muñoz González	**Iteración Asignada:** 1 iteración
Prioridad: Alta	**Tiempo Estimado:** 2 semanas
Riesgo en Desarrollo: N/A	**Tiempo Real:** 2 semanas

Descripción: A partir de una cantidad introducida por el usuario la aplicación muestra un listado ordenado con los paquetes con más descargas fallidas. El usuario podrá encontrarse con tres escenarios: el primero que inserte datos vacíos por lo que la aplicación mostrará un error de código 406, el segundo escenario cuando inserte datos incorrectos mostrará el código 405 y el último escenario es cuando ocurre el flujo normal de eventos cuando los datos son correcto.
Observaciones. • El usuario debe autenticarse • Para acceder a la URL debe utilizar el método POST. • La URL para acceder a la funcionalidad está compuesta por: La dirección donde esté publicado el servicio, el puerto 5000. Ejemplo: http//:10.53.3.98:5000/app/download/failed/apps.

Mostrar los paquetes que más fallan

Tabla 11 Historia de usuario del RF5 Mostrar los paquetes que más fallan

(Fuente: Elaboración propia)

Historia de Usuario	
Nombre del requisito: Mostrar los paquetes que más fallan	
Programador: Dareyna de la Caridad Muñoz González	**Iteración Asignada:** 1 iteración
Prioridad: Alta	**Tiempo Estimado:** 2 semanas
Riesgo en Desarrollo: N/A	**Tiempo Real:** 2 semanas
Descripción: La aplicación dado un paquete muestra la cantidad de descargas fallidas que ha tenido el mismo desde los repositorios. El usuario podrá encontrarse con tres escenarios: el primero que inserte datos vacíos por lo que la aplicación mostrará un error de código 406, el segundo escenario cuando inserte datos incorrectos mostrará el código 405 y el último escenario es cuando ocurre el flujo normal de eventos cuando los datos son correcto.	
Observaciones.	

• El usuario debe autenticarse • Para acceder a la URL debe utilizar el método POST. • La URL para acceder a la funcionalidad está compuesta por: La dirección donde esté publicado el servicio, el puerto 5000 Ejemplo: http//:10.53.3.98:5000/app/download/failed

Mostrar dirección hacia donde se descargan los paquetes.

Tabla 12 Historia de usuario del R6 Mostrar dirección hacia donde se descargan los paquetes (Fuente: Elaboración propia)

Historia de Usuario		
Número: 1	**Nombre del requisito: Mostrar dirección hacia donde se descargan los paquetes**	
Programador: Dareyna de la Caridad Muñoz González		**Iteración Asignada:** 1 iteración
Prioridad: Alta		**Tiempo Estimado:** 2 semanas
Riesgo en Desarrollo: N/A		**Tiempo Real:** 2 semanas
Descripción: La aplicación devuelve un listado con las direcciones IP (Internet Protocolo) de los destinos a los que se descargó un paquete determinado.		
Observaciones. • El usuario debe autenticarse • Para acceder a la URL debe utilizar el método GET • La URL para acceder a la funcionalidad está compuesta por: La dirección donde esté publicado el servicio, el puerto 5000 Ejemplo: http//:10.53.3.98:5000/app/apps		

Mostrar los paquetes más descargadas por fecha.

Tabla 13 Historia de usuario del RF7 Mostrar los paquetes más descargadas por fecha

(Fuente: Elaboración propia)

Historia de Usuario	
Número: 1	**Nombre del requisito: Mostrar aplicaciones más descargadas por fecha**
Programador: Dareyna de la Caridad Muñoz González	**Iteración Asignada:** 1 iteración
Prioridad: Alta	**Tiempo Estimado:** 2 semanas
Riesgo en Desarrollo: N/A	**Tiempo Real:** 2 semanas
Descripción: Dada una fecha y un número de paquetes a mostrar, la aplicación muestra un listado con los paquetes descargados exitosamente en dicha fecha. El usuario podrá encontrarse con tres escenarios: el primero que inserte datos vacíos por lo que la aplicación mostrará un error de código 406, el segundo escenario cuando inserte datos incorrectos mostrará el código 405 y el último escenario es cuando ocurre el flujo normal de eventos cuando los datos son correcto.	
Observaciones. • El usuario debe autenticarse • Para acceder a la URL debe utilizar el método POST. • La URL para acceder a la funcionalidad está compuesta por: La dirección donde esté publicado el servicio, el puerto 5000. Ejemplo: http//:10.53.3.98:5000/app/favorite/date.	

Autenticar

Tabla 14 Historia de usuario del RF8 Autenticar

(Fuente: Elaboración propia)

Historia de Usuario	
Número: 1	**Nombre del requisito: Autenticar**

Programador: Dareyna de la Caridad Muñoz González	**Iteración Asignada:** 1 iteración
Prioridad: Alta	**Tiempo Estimado:** 2 semanas
Riesgo en Desarrollo: N/A	**Tiempo Real:** 2 semanas

Descripción: Para que el usuario se autentique debe realizar los siguientes pasos:

1. Ir al campo Haerders y buscar la opción que se llama Content_application/json.
2. Ir al Body del RESTCliente y poner el usuario y la contraseña.
3. Ir al campo que se llama método y buscar la opción que se llama POST.
4. Luego el usuario accede a la URL para registrarse
5. Luego debe logarse para obtener el token, que se va encontrar en el Response del RESTCliente.
6. Luego debe ir al Header y buscar la opción Authorization y en el Bearer insertar el token que fue seleccionado en el Response anteriormente explicado.

Observaciones.

- El usuario debe autenticarse
- Para acceder a la URL debe utilizar el método POST.
- La URL para acceder a la funcionalidad está compuesta por:

  La dirección donde esté publicado el servicio, el puerto 5000 y el endpoint: app/category.

  Ejemplo: http//:10.53.3.98:5000/repository/registration.

  Ejemplo: http//:10.53.3.98:5000/repository/login.
- Cuando el cliente pone su usuario y contraseña debe estar entre comillas dobles y llaves.
- Cuando selecciona el token debe seleccionarlo sin comilla.
- Cuando el usuario ubica el token en el Baere debe estar sin los dos puntos y un espacio ante de instarlo.

Anexo 4: Encuesta a especialistas del Centro de Software Libre (CESOL)

Objetivo: evaluar la propuesta de solución desarrollada y conocer el índice de satisfacción de sus usuarios potenciales.

Especialista, le invito a responder el siguiente cuestionario con el objetivo de obtener su opinión sobre la propuesta de solución desarrollada. Solicito que exprese en sus respuestas criterios verídicos que guíen a la autora de la investigación.

1. ¿Considera importante la gestión de la información relacionada con el uso de los paquetes del repositorio de Nova?

Sí ___ No ___ No sé___

2. ¿Considera usted correcta la forma en que se analizan los log generados por el repositorio de Nova actualmente?

Sí ___ No ___ No sé___

3. ¿Considera usted factible la implementación de una API REST que permita conocer el estado de los paquetes del repositorio de Nova?

Sí ___ No ___ No sé___

4. Luego de haber mostrado los resultados de la solución refleje en qué medida le gusta la solución desarrollada.

__ Me gusta mucho

__ Me gusta más de lo que me disgusta

__ Me da lo mismo

__ Me disgusta más de lo que me gusta

__ No me gusta nada

__ No sé decir

5. ¿Qué opina usted acerca de los beneficios que traería para la universidad disponer de una solución como API REST para el uso del repositorio de Nova?

Anexo 5 Diseño de caso de prueba

Mostrar el listado de paquetes

Abreviaturas utilizadas:

CP: Caso de Prueba.

HU: Historia de Usuario.

EC: Escenario.

Tabla 15 Caso de Prueba Funcional del RF1 Mostrar listado de paquetes

(Fuente: Elaboración propia)

Caso de Prueba Funcional			
CP1_HU1		**HU_2: Mostrar el listado de paquetes**	
Responsable: Dareyna Muñoz González			
Descripción: El caso de prueba se inicia luego que el usuario entre a la aplicación. En la sección donde se inicia la URL, el usuario debe escoger el endpoint que define la URL que desea ver, ya una vez escogida la URL, en este caso muestra los paquetes descargados y la categoría asociada al paquete.			
Escenario	**Descripción**	**Respuesta del sistema**	**Flujo del sistema**
EC 1.1 Mostrar el listado de paquetes.	El usuario selecciona la URL una vez seleccionada, muestra los paquetes descargados y la categoría asociada al paquete.	Se muestra la categoría.	El usuario escoge la URL que está definida para este caso y se muestra la información.

Mostrar la cantidad de veces que se descarga un paquete con éxito

Abreviaturas utilizadas:

CP: Caso de Prueba.

HU: Historia de Usuario.

EC: Escenario.

Tabla 16 Caso de Prueba Funcional del RF3 Mostrar la cantidad de veces que se descarga un paquete con éxito

(Fuente: Elaboración propia)

Caso de Prueba Funcional

CP1_HU3		**HU_3: Mostrar la cantidad de veces que se descarga un paquete con éxito**	
Responsable: Dareyna Muñoz González			
Descripción: El caso de prueba se inicia luego que el usuario entre a la aplicación. En la sección donde se inicia la URL, el usuario debe escoger el endpoint que define la URL que desea ver, ya una vez escogida la URL, en este caso el usuario introduce un paquete que quiere mostrar.			
Escenario	**Descripción**	**Respuesta del sistema**	**Flujo del sistema**
EC 1.1 Mostrar la cantidad de veces que se descarga un paquete con éxito.	El usuario selecciona la URL una vez seleccionada, introduce el paquete a mostrar.	Se muestra la cantidad de veces que se descargó con éxito.	El usuario introduce el paquete a mostrar.
EC 1.2 Error cuando introduce datos incorrectos.	El usuario selecciona la URL una vez seleccionada, introduce datos incorrectos.	Se mostrará un error de código 405.	El usuario introduce el paquete a mostrar.
EC 1.2 Error cuando introduce datos vacíos.	El usuario selecciona la URL una vez seleccionada, introduce datos vacíos.	Se mostrará un error de código 406.	El usuario introduce el paquete a mostrar.

Mostrar listado de la cantidad de veces que un paquete tiene descarga fallida

Abreviaturas utilizadas:

CP: Caso de Prueba.

HU: Historia de Usuario.

EC: Escenario.

Tabla 17 Caso de Prueba Funcional del RF3 Mostrar la cantidad de veces que un paquete tiene descarga fallida (Fuente: Elaboración propia)

Caso de Prueba Funcional	
CP1_HU4	**HU_4: Mostrar la cantidad de veces que un paquete tiene descarga fallida**
Responsable: Dareyna Muñoz González	

Descripción: El caso de prueba se inicia luego que el usuario entre a la aplicación. En la sección donde se inicia la URL, el usuario debe escoger el endpoint que define la URL que desea ver, ya una vez escogida la URL, en este caso el usuario introduce la cantidad de paquetes que quiere mostrar.			
Escenario	**Descripción**	**Respuesta del sistema**	**Flujo del sistema**
EC 1.1 Mostrar la cantidad de veces que un paquete tiene descarga fallida.	El usuario selecciona la URL una vez seleccionada, introduce la cantidad de paquete a mostrar.	Se muestra la información de los paquetes que tienen descarga fallida.	El usuario introduce la cantidad de paquete a mostrar.
EC 1.2 Error cuando introduce datos incorrectos.	El usuario selecciona la URL una vez seleccionada, introduce datos incorrectos.	Se mostrará un error de código 405.	El usuario introduce la cantidad de paquete a mostrar.
EC 1.2 Error cuando introduce datos vacíos.	El usuario selecciona la URL una vez seleccionada, introduce datos vacíos.	Se mostrará un error de código 406.	El usuario introduce la cantidad de paquete a mostrar.

Mostrar los paquetes que más fallan

Abreviaturas utilizadas:

CP: Caso de Prueba.

HU: Historia de Usuario.

EC: Escenario.

Tabla 18 Caso de Prueba Funcional del RF5 Mostrar los paquetes que más fallan

(Fuente: Elaboración propia)

Caso de Prueba Funcional	
CP1_HU5	**HU_5: Mostrar los paquetes que más fallan**
Responsable: Dareyna Muñoz González	

Descripción: El caso de prueba se inicia luego que el usuario entre a la aplicación. En la sección donde se inicia la URL, el usuario debe escoger el endpoint que define la URL que desea ver, ya una vez escogida la URL, en este caso el usuario introduce un paquete que quiere mostrar.			
Escenario	**Descripción**	**Respuesta del sistema**	**Flujo del sistema**
EC 1.1 Mostrar la cantidad de veces que se descarga un paquete con fallo.	El usuario selecciona la URL una vez seleccionada, introduce el paquete a mostrar.	Se muestra la cantidad de veces que se descargó con fallo.	El usuario introduce el paquete a mostrar.
EC 1.2 Error cuando introduce datos incorrectos.	El usuario selecciona la URL una vez seleccionada, introduce datos incorrectos.	Se mostrará un error de código 405.	El usuario introduce el paquete a mostrar.
EC 1.2 Error cuando introduce datos vacíos.	El usuario selecciona la URL una vez seleccionada, introduce datos vacíos.	Se mostrará un error de código 406.	El usuario introduce el paquete a mostrar.

Mostrar dirección hacia donde se descargan los paquetes.

Abreviaturas utilizadas:

CP: Caso de Prueba.

HU: Historia de Usuario.

EC: Escenario.

Tabla 19 Caso de Prueba Funcional del RF6 Mostrar la dirección hacia donde se descargan los paquetes

(Fuente: Elaboración propia)

Caso de Prueba Funcional	
CP1_HU6	**HU_6: Mostrar la dirección hacia donde se descargan los paquetes**
Responsable: Dareyna Muñoz González	
Descripción: El caso de prueba se inicia luego que el usuario entre a la aplicación. En la sección donde se inicia la URL, el usuario debe escoger el endpoint que define la URL que desea ver, ya una vez escogida la URL, en este caso muestra la dirección de IP donde son descargados los paquetes.	

Escenario	Descripción	Respuesta del sistema	Flujo del sistema
EC 1.1 Mostrar la dirección de IP de los paquetes descargados.	El usuario selecciona la URL una vez seleccionada, muestra la dirección de IP de los paquetes que son descargados del repositorio.	Se la dirección de IP.	El usuario escoge la URL que está definida para este caso y se muestra la dirección de IP.

Mostrar los paquetes más descargadas por fecha.

Abreviaturas utilizadas:

CP: Caso de Prueba.

HU: Historia de Usuario.

EC: Escenario.

Tabla 20 Caso de Prueba Funcional del RF7 Mostrar los paquetes más descargado por fecha

(Fuente: Elaboración propia)

Caso de Prueba Funcional			
CP1_HU7		**HU_7: Mostrar los paquetes más descargado por fecha**	
Responsable: Dareyna Muñoz González			
Descripción: El caso de prueba se inicia luego que el usuario entre a la aplicación. En la sección donde se inicia la URL, el usuario debe escoger el endpoint que define la URL que desea ver, ya una vez escogida la URL, en este caso el usuario introduce una fecha y un número de paquetes a mostrar, la aplicación muestra un listado con los paquetes descargados exitosamente en dicha fecha e quiere mostrar.			
Escenario	**Descripción**	**Respuesta del sistema**	**Flujo del sistema**
EC 1.1 Mostrar la fecha en que se descargó un paquete.	El usuario selecciona la URL una vez seleccionada, introduce la fecha y el número de paquetes a mostrar.	Se muestra la fecha y el paquete descargado.	El usuario introduce la fecha y el número de paquete a mostrar.
EC 1.2 Error cuando introduce datos incorrectos.	El usuario selecciona la URL una vez seleccionada, introduce datos incorrectos.	Se mostrará un error de código 405.	El usuario introduce el paquete a mostrar.

EC 1.2 Error cuando introduce datos vacíos.	El usuario selecciona la URL una vez seleccionada, introduce datos vacíos.	Se mostrará un error de código 406.	El usuario introduce el paquete a mostrar.

Autenticar

Abreviaturas utilizadas:

CP: Caso de Prueba.

HU: Historia de Usuario.

EC: Escenario.

Tabla 21 Caso de Prueba Funcional del RF8 Autenticar

(Fuente: Elaboración propia)

Caso de Prueba Funcional			
CP1_HU8		**HU_8: Autenticar**	
Responsable: Dareyna Muñoz González			
Descripción: El caso de prueba se inicia luego que el usuario entre a la aplicación. En la sección donde se inicia la URL, el usuario debe escoger el endpoint que define la URL que desea ver, ya una vez escogida la URL, en este caso la autenticación.			
Escenario	**Descripción**	**Respuesta del sistema**	**Flujo del sistema**
EC 1.1 La aplicación permite introducir el usuario y la contraseña.	El usuario selecciona la URL una vez seleccionada, introduce el usuario y la contraseña.	Se muestra el token.	El usuario introduce el token en el campo Authorization_Bearer.
EC 1.2 Error cuando introduce datos incorrectos.	El usuario selecciona la URL una vez seleccionada, introduce datos incorrectos.	Se mostrará un error de código 405.	El usuario introduce el paquete a mostrar.
EC 1.2 Error cuando introduce datos vacíos.	El usuario selecciona la URL una vez seleccionada, introduce datos vacíos.	Se mostrará un error de código 406.	El usuario introduce el paquete a mostrar.