Universidad de las Ciencias Informáticas

Facultad 1

Módulo para la administración remota del servicio telemático Nginx en la plataforma Nova ARST

Trabajo de Diploma para optar por el título

de Ingeniero en Ciencias Informáticas

Autor:

Leovaldo Pérez Rojas

Tutores:

Yurisbel Vega Ortiz

Lexys Manuel Días Alonso

La Habana, junio de 2022
"Año 63 de la Revolución"

Agradecimientos

A lo largo de mi carrera he conocido a muchas personas que me han brindado su apoyo y sin las cuales quizás no hubiese logrado llegar a redactar estas palabras. También he podido reafirmar que tengo la familia más especial que se puede tener y que sin lugar a dudas si volviera a nacer y tuviera la posibilidad de escoger una familia, elegiría esta porque con sus buenos y malos momentos me ha demostrado y enseñado que la familia es lo primero y que siempre está ahí no importa cuán complicada sea la situación.

Es por eso que quiero agradecer a todos los ***amigos*** *que he conocido en estos años, a los que por cualquier razón no siguen a mi lado físicamente, pero me siguen bridando su apoyo y a los que aún siguen luchando por el mismo sueño que yo, el de lograr el título de Ingeniero en Ciencias Informáticas. En especial quisiera agradecer a* ***Ernesto*** *y* ***Neysa****, mis padres aquí en la UCI con los cuales pasé mis mejores momentos en la Universidad y ojalá estos nunca terminen.*

Quiero agradecer también a todos y cada uno de los ***profesores*** *de los cuales aprendí mucho no solo de las materias que cada uno impartía sino también de la vida en general.*

También quiero agradecer a Yurisbel y a Lexys que son mis tutores y has sido un gran apoyo en este proceso tan complicado.

También me gustaría agradecer a ***Nurisel*** *quien fue mi primer tutor de la tesis y aunque no haya terminado el proceso junto a mí fue de gran ayuda al principio y aportó mucho a esta investigación.*

A todos muchas gracias.

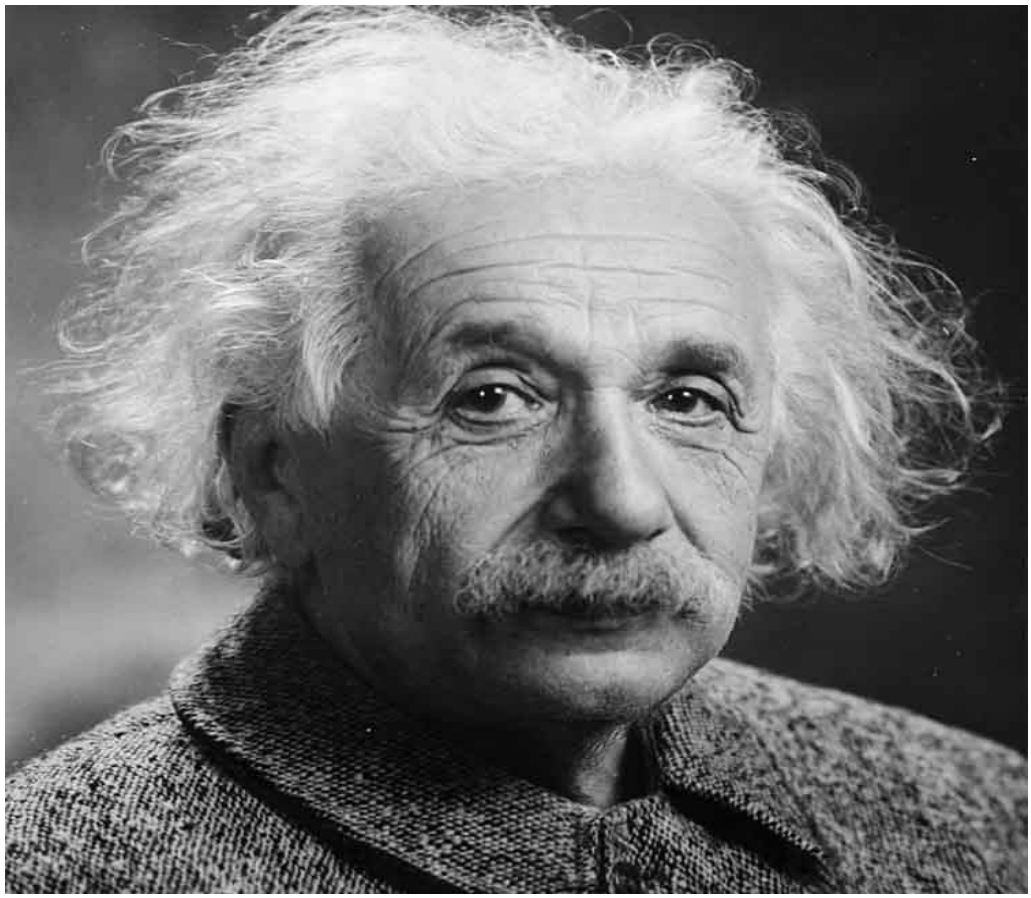
A mis ***padres*** *por siempre ser un apoyo en todas mis decisiones y por satisfacer todas mis necesidades y caprichos.*

A mi ***hermana*** *y mi* ***sobrina*** *porque son muy especiales para mí y por siempre estar ahí cuando las he necesitado.*

A mis ***abuelas****, mis tías y mi prima porque siempre que las he necesitado me han apoyado.*

En general a toda mi ***familia*** *por su apoyo incondicional en todo momento.*

A todos ustedes con todo mi amor y dedicación.



"No podemos resolver nuestros problemas con el mismo pensamiento que usamos cuando los creamos."

Albert Einstein

Declaración de autoría

Declaro por este medio que yo **Leovaldo Pérez Rojas**, con carné de identidad **98122308283** soy el autor principal del trabajo titulado "**Módulo para la administración remota del servicio telemático Nginx en la plataforma Nova ARST**" y autorizo a la Universidad de las Ciencias Informáticas a hacer uso de la misma en su beneficio, así como los derechos patrimoniales con carácter exclusivo.

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

_____

Leovaldo Pérez Rojas

Autor

_____

Yurisbel Vega Ortiz

Tutor

_____

Lexys Manuel Días Alonso

Tutor

Resumen

Cuba desde hace varios años se encuentra inmerso en el proceso de inmigración hacia software libre. La Universidad de la Ciencias Informáticas (UCI) es una de las instituciones que impulsan este proceso. En ella se encuentra el departamento de Servicios Integrales de Migración, Asesoría y Soporte (SIMAYS) donde se desarrolla una herramienta que permite administrar servicios telemáticos de forma remota (Nova ARST).

Actualmente la configuración del servidor web Nginx se realiza de forma manual a través de una interfaz de consola lo que trae consigo una pérdida de tiempo en el despliegue del servicio. Debido a esta problemática se realizó la presente investigación con el objetivo de desarrollar una aplicación que permita realizar la configuración del servidor web Nginx. Para ello se analizaron herramientas con las que se administra este servicio en diferentes tecnologías para determinar si alguna daba solución de forma total o parcial al problema detectado. En la presente investigación se exponen las tecnologías y lenguajes a utilizar en la implementación del módulo y la definición de los elementos conceptuales necesarios para un mayor entendimiento del servicio. Para regir el desarrollo de la solución se utilizó la metodología AUP-UCI, adaptada a los procesos de la Universidad. Para la validación del módulo desarrollado se ejecutaron diferentes pruebas de software para verificar la calidad y correcto funcionamiento del mismo. Como resultado de la investigación se obtuvo un módulo que permite la configuración del servidor web Nginx desde la plataforma Nova ARST.

Palabras Clave: configuración, Nova ARST, servidor web Nginx

Abstract

Cuba for several years has been immersed in the process of immigration towards free software. The University of Informatics Sciences (UCI) is one of the institutions promoting this process. The Department of Comprehensive Migration, Consulting and Support Services (DCMCSS) is located there, where a tool is being developed that allows telematics services to be managed remotely (Nova ARST).

Currently the configuration of the Nginx web server is done manually through a console interface, which results in a loss of time in the deployment of the service. Due to this problem, the present investigation was carried out with the objective of developing an application that allows the configuration of the Nginx web server. For this, tools with which this service is administered in different technologies were analyzed to determine if any of them gave a total or partial solution to the problem detected. In the present investigation, the technologies and languages to be used in the implementation of the module and the definition of the conceptual elements necessary for a greater understanding of the service are exposed. To govern the development of the solution, the AUP-UCI methodology was used, adapted to the processes of the University. For the validation of the developed module, different software tests were carried out to verify its quality and correct operation. As a result of the investigation, a module was obtained that allows the configuration of the Nginx web server from the Nova ARST platform.

Keywords: setting, Nova ARST, web server Nginx

Índice de contenido

Índice de tablas

Índice de figuras

Introducción

El tráfico de web es el responsable de un buen porcentaje del tráfico de Internet. Esta tendencia ha ido creciendo gradualmente desde que apareció la web, y hoy día el tráfico HTTP (del inglés *Hipertext Transfer Protocol*) predomina respecto al resto de los protocolos, y hay una gran población de usuarios que pueden generar una cantidad inmensa de peticiones si el contenido es interesante. La organización de un servicio web conectado a Internet requiere tener en cuenta las características de la demanda que pueda tener que atender. Por otro lado, la web es un servicio muy reclamado por todo tipo de organizaciones para publicar información, por lo que el crecimiento del número de servidores web en Internet ha sido exponencial (Vilajosana & Navarro, 2019).

Los servidores web son un componente de los servidores que se mantienen a la espera de peticiones por parte de un cliente (un navegador web o un programa que hace una llamada a un servicio web). Cuando el servidor recibe una petición, responde adecuadamente mediante una página web que se exhibirá en el navegador, o bien mostrará el mensaje de error correspondiente (Vilajosana & Navarro, 2019).

La comunicación entre un servidor web y sus clientes se basa en el protocolo HTTP o en su variante codificada HTTPS (del inglés *Hipertext Transfer Protocol Secure*). El servidor web está permanentemente en espera de una solicitud de información. Además, debe tenerse en cuenta que toda computadora, teléfono inteligente o tableta tiene una dirección IP (del inglés *Internet Protocol*) única e irrepetible que lo identifica de otro dispositivo en la red, así es como el servidor web envía la información exacta que el usuario está esperando. La solicitud debe hacerse además desde un navegador que es el encargado de enviar la misma desde una dirección IP hacia la dirección IP del servidor que aloja los archivos requeridos. Luego de este proceso es el servidor web nuevamente el encargado de interpretar las líneas de código y realizar una búsqueda de la información solicitada para enviarla al navegador cuya dirección IP fue la solicitante (de Souza, 2019).

Actualmente son numerosos los servidores web, dentro de los que se encuentran Microsoft-IIS, Apache 2, Nginx, LiteSpeed, Google, Sun Java System, Lighttpd, IBM Servers y Yahoo Traffic Server. Algunos son más usados que otros debido a que poseen características distintivas que marcan la diferencia en cuanto a su selección. La información sobre el uso de servidores web es proporcionada por algunos sitios especializados en ofrecer estadísticas de diferentes tecnologías en la web (Palma, 2019). Apache 2 y Nginx son los dos servidores web de código abierto más usados. Según el sitio Netcraft, hasta enero del 2022 el

uso de Apache es de 24% y el de Nginx 32% (Netcraft, 2022); por otra parte, según el reporte realizado por W3Techs hasta el 21 de febrero del 2022, el uso de Apache es de 30,7% y de Nginx 33,1% (W3Techs, 2022).

Nginx es un servidor web multiplataforma, se ejecuta en Windows, GNU/Linux, Unix, BSD, Mac OS X y Solaris. Posee una arquitectura basada en eventos, en esta arquitectura las solicitudes se aceptan utilizando *sockets* asíncronos y no se procesan en hilos separados para reducir la memoria y la sobrecarga de la CPU (del inglés *Central Processing Unit*). Tiene soporte para *hosts* virtuales de forma nativa. Nginx admite FastCGI (del inglés *Common Gateway Interface*), uWSGI (del inglés *Web Server Gateway Interface*) y SCGI (así como HTTP proxying) a través de módulos que se incluyen por defecto en tiempo de compilación. Presenta una clara ventaja cuando se sirve contenido estático, por otra parte, requiere un esfuerzo extra para que funcione con PHP, Python, Perl y otros lenguajes. El sistema de módulos es estático, los módulos están incorporados y deben incluirse en tiempo de compilación (Nedelcu, 2015) (Soni, 2016).

En Cuba también se evidencia el uso de Nginx en los servidores de las instituciones, pues para lograr la soberanía tecnológica en el país, se realiza un proceso paulatino de migración hacia tecnologías de software libre y código abierto. En cuanto al sistema operativo en los servidores de las instituciones, la alternativa libre a instalar es Nova Servidores, una de las variantes de la distribución cubana de GNU/Linux Nova. Esta distribución es desarrollada por el Centro de Software Libre (CESOL) de la Universidad de las Ciencias Informáticas (UCI), e incluye además las variantes Nova Escritorio y Nova Ligero.

Con el propósito de facilitar la administración de los servicios telemáticos, en CESOL se desarrolló la plataforma Nova ARST. Esta plataforma permite la administración remota desde una estación de trabajo con la distribución Nova Escritorio, de los servicios telemáticos ofrecidos por el servidor. El empleo de esta plataforma facilita a los administradores de redes, no tener que desplazarse hasta la ubicación física de los servidores de la institución ya que como se menciona anteriormente esta se podrá realizar de forma remota. Sin embargo, actualmente la plataforma no permite la configuración de Nginx, dificultándose la administración de este servicio, debido a que actualmente la configuración se realiza a través de la consola de Linux con líneas de comando, esto provoca en muchas ocasiones que surjan problemas provocados por el error humano, además, la configuración manual del servicio trae consigo pérdida de tiempo y un mayor esfuerzo de los encargados de la configuración del servidor provocado por las numerosas líneas de comando que deben de memorizar.

Por lo descrito anteriormente se identifica como **problema de investigación** la siguiente interrogante ¿Cómo administrar el servidor web Nginx de forma remota desde la herramienta Nova ARST?

Para la presente investigación se tiene como **objeto de estudio** la administración del servidor web Nginx en sistemas GNU Linux y el **campo de acción** está enmarcado en las herramientas informáticas para la administración del servidor web Nginx.

El **objetivo general** de este trabajo de investigación es desarrollar un módulo para la administración remota del servidor web Nginx en la plataforma Nova ARST.

Para desarrollar la investigación se plantean los siguientes **objetivos específicos:**

1. Elaborar el marco teórico de la investigación sobre la administración del servidor web Nginx.
2. Diseñar un módulo para la administración remota del servidor web Nginx en la plataforma Nova ARST.
3. Implementar un módulo para la administración remota del servidor web Nginx en la plataforma Nova ARST.
4. Evaluar el módulo para la administración remota del servidor web Nginx en la plataforma Nova ARST.

Para realizar la investigación se definen las siguientes tareas de investigación:

1. Realización del diseño teórico metodológico y redacción de la introducción.
2. Caracterización de las herramientas de administración de la configuración y selección de la que será utilizada, identificando las configuraciones que se pueden realizar sobre Nova.
3. Estudio de las tecnologías de desarrollo y selección de las que serán utilizadas.
4. Caracterización de la metodología AUP-UCI y estudio de la norma ISO-690.
5. Definición de la propuesta de solución y requisitos funcionales y no funcionales a implementar.
6. Realización de la especificación de las historias de usuarios.
7. Diseño de los prototipos de interfaz de usuario.
8. Realización del diseño del módulo para la plataforma Nova ARST.
9. Descripción de la arquitectura del módulo para la plataforma Nova ARST.
10. Implementación del módulo para la plataforma Nova ARST.

11. Diseño y aplicación de los casos de pruebas.

Se plantean como **preguntas científicas**:

1. ¿Cuáles son los presupuestos teóricos que fundamentan el proceso de administración del servidor web Nginx?
2. ¿Qué características debe tener el módulo que facilite la administración del servidor web Nginx para la plataforma Nova ARST?
3. ¿Qué componentes se deben desarrollar en el módulo para la administración remota del servidor web Nginx en la plataforma Nova ARST?
4. ¿Qué métodos o técnicas aplicar para evaluar la calidad del módulo para la administración remota del servidor web Nginx en la plataforma Nova ARST?

Para el cumplimiento del objetivo propuesto y de las tareas de investigación se emplean métodos teóricos y empíricos de la investigación científica. **Los métodos teóricos** que se utilizan son:

Analítico-Sintético: El análisis y la síntesis funcionan como una unidad dialéctica y de ahí que al método se le denomine analítico-sintético. El análisis se produce mediante la síntesis de la propiedades y características de cada parte del todo, mientras que la síntesis se realiza sobre la base de los resultados del análisis (Rodríguez & Pérez, 2017). Se pone en práctica para el estudio de diferentes fuentes bibliográficas y de los elementos del problema planteado, profundizando en las características del servidor web Nginx.

Modelación: La modelación es una praxis cognitiva que supone la construcción de una representación mental del objeto de la modelización (Rodríguez & Roggero, 2014). En la presente investigación se utilizó para la creación de abstracciones como los diagramas, con el objetivo de explicar la realidad.

Los métodos empíricos que se utilizan son:

Entrevista: La entrevista es una técnica de gran utilidad en la investigación cualitativa para recabar datos; se define como una conversación que se propone un fin determinado distinto al simple hecho de conversar. Se utilizó para investigar acerca de la plataforma Nova ARST, como se aprecia en el Anexo 1 y para conocer cómo se configura el servidor web nginx en la UCI como se refleja en el Anexo 2.

Observación para realizar un análisis de las diferentes soluciones que se utilizan tanto en la Universidad de las Ciencias Informáticas como a nivel mundial y resuelven de forma parcial la configuración de forma remota del servidor web Nginx.

El presente documento está estructurado en introducción, tres capítulos, conclusiones generales, referencias bibliográficas, anexos y glosario de términos.

En el primer capitulo “Fundamentación teórica sobre el proceso de administración del servidor web Nginx” se realiza el estudio del estado del arte donde se hace referencia a los principales conceptos asociados al objeto de estudio en la presente investigación. Se realiza un análisis bibliográfico basándose en los métodos teóricos expuestos anteriormente, que permite caracterizar el servidor web Nginx. Además, se describe la plataforma Nova ARST, su arquitectura y se hace un estudio de otras herramientas homólogas que son utilizadas para resolver parte del problema planteado. También se describe brevemente la metodología de desarrollo a emplear y se definen las herramientas y tecnologías que se utilizan para el desarrollo del módulo.

En el segundo capítulo “Análisis y diseño del módulo Nginx en la plataforma Nova ARST”, se define la propuesta de solución, se exponen las características principales del módulo y se especifican las funcionalidades a implementar. Se presentan las Historias de Usuario correspondientes a las funcionalidades y los prototipos de interfaz de usuario, así como los diagramas de clases con estereotipos web necesarios para un mayor entendimiento de cómo se desarrolla el módulo, se define la arquitectura del módulo y se describen los patrones de diseño que se utilizan en la implementación.

En el tercer capítulo “Implementación y validación del módulo para la administración remota del servidor web Nginx desde la plataforma Nova ARST.” se presenta el diagrama de despliegue del módulo, así como los estándares de codificación que se utilizan en cada uno de los lenguajes empleados en el desarrollo de dicho módulo, también se realiza la implementación de la propuesta de solución y se describen, diseñan, realizan y controlan los casos de prueba que se aplicarán sobre el módulo desarrollado.

Capítulo 1: Fundamentación teórica sobre el proceso de administración del servidor web Nginx

En este capítulo se realiza un estudio sobre los servidores web, enfocándose en el servidor web Nginx. Además, se describe la plataforma ARST, así como su arquitectura y se exponen las consideraciones a tener en cuenta para integrar un módulo en esta plataforma. También se detallan las herramientas y tecnologías que se utilizan en el desarrollo del módulo para la administración remota del servidor web Nginx y la metodología de desarrollo que se utiliza.

1.1 Conceptos fundamentales

En el presente epígrafe se expone las principales definiciones o conceptos para una mejor comprensión de la presente investigación.

Como se define en la introducción el objetivo fundamental de esta investigación es desarrollar un módulo para la administración del servidor web Nginx. En programación, un **módulo** es un fragmento de un programa que se desarrolla de forma independiente del resto del programa. Esta independencia hace posible un mecanismo de compilación por separado que limita la complejidad del programa que se está desarrollando (Gallardo & García, 2007).

El módulo a desarrollar será integrado en la plataforma Nova ARST. Una **plataforma informática** es un conjunto de sistemas o módulos que se integran con el fin de realizar uno o varios procesos de manera automatizada utilizando los recursos de *hardware* de un ordenador (Ninasunta & Pisco, 2021). Una de las características que definen a la plataforma Nova ARST es que permite la administración remota de los servicios telemáticos que ella brinda. Un software de **administración remota** ofrece la posibilidad de acceder y controlar, total o parcialmente, computadoras desde cualquier parte del mundo. Lo hace a través de *Internet* o una red local para ejecutar diferentes actividades deseadas entre un usuario y otro. Este tipo de herramientas se basan en la tecnología de cliente/servidor. El servidor se ejecuta en la computadora que es controlada, que a su vez recibe instrucciones del cliente que es instalado como *host* remoto (Velásquez, 2021). Por último, los **servicios telemáticos** son los servicios destinados a la difusión, almacenamiento y

tratamiento de la información, en cualquiera de sus formas, ya sea voz, video, datos o una combinación de todas ellas (Gonzáles L. , 2019), tales como correo electrónico, DNS, DHCP, proxy, mensajería instantánea y servidor web. En el epígrafe siguiente se aborda el tema de los servidores web y su arquitectura.

1.2 Servidor web

Un servidor web es una aplicación que responde a solicitudes provenientes de navegadores web, proporcionando recursos solicitados a través del protocolo HTTP o de manera segura a través del protocolo HTTPS (Mejía, Gonzáles, & España, 2018). Un servidor web brinda las respuestas HTTP, generalmente en forma de páginas web que contienen contenido estático (texto, imágenes, etcétera) y dinámico (*scripts*) (Kunda, Chihana, & Sinyinda, 2017).

Según (Palma, 2013) existen varios tipos de servidores web dependiendo de su arquitectura, entre los que se encuentran:

- **Servidores basados en procesos:** Se basan en la obtención de paralelismo mediante la duplicación del proceso de ejecución. Es el predecesor de los demás diseños. El más simple de este tipo de diseños es en el que el proceso principal con la llegada de una nueva conexión se duplica creando una copia exacta que atenderá la misma.
- **Servidores basados en hilos:** le son aplicables los conceptos básicos respecto al funcionamiento de un servidor basado en procesos, heredando muchas de sus características, entre ellas la de la simplicidad de su diseño e implementación. Las principales diferencias de los dos modelos residen en el propio concepto de hilo. Tiene la ventaja que la creación de un hilo es menos costosa que la de un proceso. Varios hilos de un mismo proceso comparten el mismo espacio de memoria, propiciando que puedan compartir datos entre ellos, sin embargo, esto implica un riesgo de seguridad.
- **Servidores basados en *sockets* no bloqueantes o dirigidos por eventos:** que basan su funcionamiento en la utilización de lecturas y escrituras asincrónicas sobre *sockets*. Utilizan una llamada al sistema que examine el estado de los *sockets* operativos que implementan una o más funciones de examen de *sockets*. Estas funciones tienen como objetivo inspeccionar el estado de un grupo de *sockets* asociados a cada una de las conexiones. Este tipo de servidores a pesar de

caracterizarse por su velocidad, tienen la desventaja que la concurrencia es simulada, existiendo un solo proceso y un solo hilo, desde el cual se atienden todas las conexiones.

El servidor web Nginx presenta una arquitectura basada en *sockets* no bloqueantes o dirigidos por eventos. En el siguiente epígrafe se hace un acercamiento a las características, arquitectura y configuración de dicho servidor.

1.3 Caracterización del servidor web Nginx

Nginx fue creado por Igor Sysoev. Comenzó a desarrollarse alrededor del año 2002 y se lanzó oficialmente en 2004. El equipo de desarrollo de Nginx lo describe como "*un servidor HTTP libre, código abierto, de alto rendimiento y proxy inverso, así como un servidor proxy IMAP/POP3*". Nginx es conocido por su alto rendimiento, estabilidad, amplio conjunto de funciones, configuración simple y bajo consumo de recursos. Nginx ha resuelto el problema de rendimiento y esa es una de las principales razones de todos los elogios y premios que conlleva. Es extremadamente rápido y brilla incluso bajo mucha carga. Nginx en realidad nunca comenzó como un servidor para lenguajes dinámicos como PHP. Nginx ha evolucionado desde un simple experimento con la idea de resolver el problema C10K (Soni, 2016).

1.3.1 Arquitectura del servidor web Nginx

Nginx es un servidor asincrónico construido buscando solucionar los problemas de concurrencia que experimentaban ciertos sitios. El algoritmo desarrollado para este servidor es muchísimo más eficiente y consume menos recursos. Nginx genera procesos *worker*, cada uno de los cuales puede manejar miles de conexiones. Se puede lograr esto debido a la implementación de un mecanismo de bucle rápido que busca y procesa eventos continuamente. Cada conexión manejada por el *worker* es dispuesta dentro del bucle de eventos, donde vive con otras conexiones. Los eventos dentro de este bucle se procesan de forma asincrónica, permitiendo que el trabajo sea manejado de forma no bloqueante. Cuando una conexión se cierra, se elimina del bucle. Esta disposición asincrónica y mono hilos ayuda a que incluso con recursos limitados Nginx sea bastante rápido en cuanto al procesamiento de conexiones. Por consiguiente, el uso de CPU (del inglés *Central Processing Unit*) y memoria tiende a bajar debido a la eficiencia en el manejo de conexiones (Burgos, 2016).

1.3.2 Características de Nginx

- **Multiplataforma:** inicialmente creado para funcionar en sistemas operativos Unix, más tarde apareció una versión compatible con *Windows*, y quedó listo para trabajar en Linux, Unix, Windows, MacOS, Solaris y BSD (Nedelcu, 2015).
- **Sistema de módulos estático:** los módulos no pueden ser cargados de forma dinámica, deben ser incluidos en la compilación. Además, no se pueden desactivar en tiempo de ejecución debido a que están compilados e integrados en el binario principal (Nedelcu, 2015).
- **Soporte de virtual *host*:** posee soporte para virtual *host* basados en IP y/o virtual *host* basados en nombre (Palma, 2019).
- **Proxy inverso:** el trabajo de un servidor proxy es enviar el pedido por adelantado y pasarlo al servidor proxy, que también se conoce como un servidor en sentido ascendente. El servidor ascendente procesa la solicitud y envía la respuesta nuevamente al servidor Nginx, que además retransmite la respuesta a los clientes que hicieron la solicitud. Nginx puede ser utilizado como proxy inverso con opciones de caché. Nginx puede hacer fácilmente el trabajo de *front-end* y enrutar el tráfico según sus requisitos al *back-end* utilizando sus capacidades de proxy inverso (Soni, 2016).
- **Requerimientos de *hardware*:** en Nginx se genera una carga de CPU y un consumo de memoria mucho más ligeros (Nedelcu, 2015).

En el siguiente epígrafe se da a conocer la configuración del servidor web Nginx y sus principales directivas de configuración.

1.3.3 Configuración de Nginx

El servidor web Nginx se instala a través de la consola de Linux, en cualquier sistema operativo basado en GNU/Linux en la ruta */etc/nginx*. El archivo de configuración de Nginx se llama **nginx.conf** y generalmente estará bajo el directorio */usr/local/nginx/conf*, */etc/nginx* o */usr/local/etc/nginx*. El funcionamiento de Nginx y sus módulos depende de dicho archivo. Los directorios principales de configuración de los virtual *host* son *sites-available* (donde se crean los archivos de configuración específicos de cada virtual *host* disponible) y *sites-enabled* (en el que se crean los enlaces simbólicos a los archivos de configuración que se hayan creado en *sites-available*).

Según (Nedelcu, 2015) Nginx tiene dos tipos de directivas:

- Directivas simples: las cuales están compuestas por el nombre de la directiva junto con sus parámetros separados por espacio, terminando con un punto y coma
- Directivas de bloque: tiene la misma estructura de una directiva simple, pero en lugar de un punto y coma proveen un conjunto de instrucciones rodeadas de llaves

También se definen los contextos. Una directiva de bloque pasa a ser un "contexto" cuando tiene otras directivas entre las llaves (por ejemplo: *events*, *http*, *server* y *location*).

Debido a que los contextos pueden estar anidados, Nginx ofrece un nivel de herencia de directivas. Una regla general de Nginx es que, si una directiva es válida en varios alcances anidados, una declaración en un contexto más amplio será pasado a cualquier contexto hijo como valor predeterminado. Los contextos hijos pueden sobrescribir estos valores a su voluntad. Las directivas solo pueden ser usadas en los contextos para los cuales están diseñadas.

Módulos principales

Los módulos principales o "*base modules*" definen la configuración básica de Nginx. Se incorporan automáticamente a Nginx en tiempo de compilación. Estas directivas y bloques están disponibles siempre. Hay tres tipos de módulos principales (Nedelcu, 2015).

El módulo ***core*** contiene directivas y características esenciales para el funcionamiento de Nginx. La mayoría de sus directivas deben estar colocadas en la raíz (la parte superior del archivo de configuración). Según (Nedelcu, 2013) las que se muestran a continuación son algunas de las directivas de este módulo.

Tabla 1 Directivas del módulo core

(Fuente: Elaboración propia)

Comando y contexto	Descripción
daemon	Valores aceptados: *on/off* Sintaxis: *daemon on*; Valor por defecto: *on* Habilita o deshabilita el modo *daemon*. Si lo desactiva, el programa no se iniciará en segundo plano; permanecerá en primer plano cuando se lanza desde la terminal. Esto puede ser útil para depuración, en situaciones en las que necesita saber qué causa que Nginx deje de funcionar y cuándo.
user	Sintaxis: *user username groupname*; *user username*; Permite definir la cuenta de usuario, y opcionalmente, el grupo utilizado para iniciar los procesos *worker*. Se deben utilizar usuarios y grupos con privilegios limitados por razones de seguridad.
worker_processes	Sintaxis: Entero o auto *worker_processes* 4; Valor por defecto: 1 Define la cantidad de procesos *worker*. Si un proceso está bloqueado por alguna razón, las solicitudes futuras pueden ser servidas por otro proceso *worker*. Si se utiliza el valor “auto” Nginx seleccione el valor apropiado (por defecto es el número detectado de núcleos de CPU).
error_log Contexto: *main, http, server and location*	Sintaxis: *error_log /file/path level*; Valor por defecto*: logs/error.log error* Indica la ubicación de los archivos de registro de eventos.

	Si se desea desactivar el registro de errores, envíe el log a la siguiente ruta: */dev/null.*
pid	Sintaxis: Directorio *pid logs/nginx.pid*; Valor por defecto: Definido en el momento en que se compila Es la ruta al archivo donde se almacenará el PID para Nginx.
env	Sintaxis: *env MY_VARIABLE*; *env MY_VARIABLE=my_value*; Esta directiva posibilita definir las diferentes variables y los entornos a los que pertenecen.
master_process	Valores aceptados: *on/off* Sintaxis: *master_process on*; Valor por defecto: *on* Determina si se inician los procesos de trabajo.

El módulo ***events*** está contenido en el módulo principal. Se usa para establecer las opciones que afectan la forma en que Nginx maneja las conexiones a nivel general. Nginx utiliza un modelo de procesamiento de conexiones basado en eventos, de modo que las directivas definidas en este módulo determinan cómo los procesos *worker* deben manejar las conexiones (Nedelcu, 2015).

Tabla 2 Directivas del módulo events

(Fuente: Elaboración propia)

Comando y contexto	**Descripción**
accept_mutex	Valores Aceptados: *on/off* Valor por defecto*: on* Sintaxis: *accept_mutex on*; Habilita o deshabilita el uso de una exclusión mutua para abrir puertos de escucha.
accept_mutex_delay	Valor por defecto: 500 milisegundos

	Sintaxis: Entero (tiempo) *accept_mutex_delay 500 ms*; Define la cantidad de tiempo que un proceso de trabajo debe esperar antes de adquirir el recurso de nuevo. Este valor no se utiliza si la directiva *accept_mutex* está desactivada.
worker_connections	Valor por defecto: ninguno Sintaxis: Entero *worker_connections* 1024; Define la cantidad de conexiones que un proceso de trabajo puede tratar simultáneamente.
debug_connections	Sintaxis: Dirección IP o bloque CIDR *debug_connections* 172.63.155.21; *debug_connections* 172.63.155.0/24; Valor por defecto: ninguno Escribe registros detallados para clientes que coincidan con esta dirección IP o bloque de direcciones. La información de depuración se almacena en el archivo especificado con la directiva *error_log*, habilitado con el nivel de depuración. Nota: Nginx debe compilarse con el modificador --debug en orden para habilitar esta función.

El módulo ***configuration*** permite inclusiones de archivos con la directiva *include*. Las directivas pueden estar dispuestas en cualquier lugar del archivo de configuración y aceptan un único parámetro: la ruta del archivo (Burgos, 2016).

include /archivo/ruta.conf;

include sitios/.conf;*

Configuración del *virtual host*

El *virtual host*, o servidor virtual, es una forma de alojamiento web que permite que varias páginas web puedan funcionar en una misma máquina. Hay dos tipos de *virtual host*:

- Los que se basan en direcciones IP, donde cada página web tendrá una IP diferente.
- Los que se basan en nombres de dominio, donde en una sola dirección IP funcionan varias páginas web.

Aunque el navegador tendrá que diferenciar el tipo de virtual *host* a la hora de gestionar la petición, la elección de una u otra no tiene ningún efecto para el usuario. La configuración de los virtual *host* en el servidor Nginx se realiza en el directorio */etc/nginx*, en el mismo se encuentran las carpetas *sites-available, sites-enabled, modules-available, modules-enabled,* etcétera. y los enlaces simbólicos se crean manualmente. (Linux, Proyectos, 2018).

En la siguiente tabla se muestran las principales directivas que se utilizan para configurar un virtual *host* en Nginx.

Tabla 3 Directivas de configuración de los virtual host

(Fuente: Elaboración propia)

Directiva	**Descripción**
listen Contexto: *server*	Especifica la dirección IP y/o el puerto que utilizará el conector de escucha para servir el sitio web. Los sitios generalmente se sirven en el puerto 80 (el valor predeterminado) a través de HTTP, o 443 a través de HTTPS. Sintaxis: *listen* [dirección IP] [: puerto] [opciones adicionales];
default_server Contexto: *server*	Especifica que este bloque de servidor se utilizará como predeterminado.
root Contexto: *http, server, location, if*	Define la ruta del documento *root*, contiene los ficheros que se mostraran a los clientes. Sintaxis: Dirección de un fichero Valor por defecto: html
server_name	Asigna uno o más nombres de *host* al bloque del servidor. Cuando Nginx recibe una solicitud HTTP, hace coincidir el encabezado *Host*

Contexto: *server*	de la solicitud con todos los bloques del servidor. Se selecciona el primer bloque de servidor que coincida con este nombre de *host*. Sintaxis: *server_name hostname1* [*hostname2*…];

1.3.4 Comandos para manejar el servicio

Tabla 4 Comandos principales para el servidor web Nginx

(Fuente: Elaboración propia)

Comando	Descripción
$ sudo apt get install nginx	Para instalar el servidor web Nginx con el administrador de paquetes de distribución predeterminado.
$ nginx -V	Para ver la versión y configurar las opciones
$ sudo nginx -t	Antes de iniciar realmente el servicio Nginx, puede verificar si su sintaxis de configuración es correcta. Esto es especialmente útil si ha realizado cambios o agregado una nueva configuración a la estructura de configuración existente.
$ sudo nginx -T	Puede probar la configuración de Nginx y salir usando el indicador -T como se muestra.

$ sudo systemctl start nginx #systemd
OR
$ sudo service nginx start #sysvinit
	Para iniciar el servicio Nginx, ejecute el siguiente comando Tenga en cuenta que este proceso puede fallar si la sintaxis de configuración no es correcta.

$ sudo systemctl enable nginx #systemd
OR
$ sudo service nginx enable #sysv init
	El comando anterior solo inicia el servicio mientras tanto, para poder habilitar el inicio automático en el momento del arranque, ejecute el siguiente comando.

$ sudo systemctl restart nginx #systemd	Para reiniciar el servicio Nginx, una acción que se detendrá y luego iniciará el servicio.

OR $ sudo service nginx restart #sysv init	

$ sudo systemctl status nginx #systemd
OR
$ sudo service nginx status #sysvinit
	Puede verificar el estado del servicio Nginx de la siguiente manera. Este documento muestra la información de estado del tiempo de ejecución sobre el servicio.

$ sudo systemctl reload nginx #systemd
OR
$ sudo service nginx reload #sysvinit
	Para decirle a Nginx que vuelva a cargar su configuración.

$ sudo systemctl stop nginx #systemd
OR
$ sudo service nginx stop #sysvinit
	Para detener el servicio Nginx

$ systemctl -h nginx	Para obtener una guía de referencia fácil de todos los comandos y opciones de Nginx.

1.4 Descripción de la plataforma Nova ARST

La plataforma Nova ARST es una nueva plataforma que se está desarrollando en el Centro de Software Libre (CESOL) de la UCI para la distribución cubana de GNU/Linux Nova Servidores. La ventaja que ofrece esta nueva plataforma es que permite la configuración de cualquiera de los módulos que se integraran en la misma, de una manera mucho más fácil e intuitiva para cualquier usuario, aunque este no presente mucha experiencia en la configuración de los mismos. Otra ventaja que ofrece esta plataforma es que se puede acceder a ella desde cualquier estación de trabajo sin necesidad de desplazarse a donde esté el servidor físicamente. Esta plataforma utiliza una arquitectura n-capas. La arquitectura n-capas consiste en dividir un software en varias partes lógicas, ya sean módulos, paquetes o capas. Esto ofrece la posibilidad de

comprender fácilmente su filosofía y distribuir las tareas que ejecuta. Por ello la comunidad del software desarrollo la noción de una arquitectura de varios niveles y entre las más difundidas se encuentra la *"tres capas"*. La Arquitectura en Tres Capas divide la aplicación en tres partes lógicas, con un grupo de interfaces perfectamente definidas (Montes de Oca & Brito, 2012). Para facilitar comunicación entre el servidor y los clientes se diseñó utilizando una API Rest. **API REST** es una interfaz entre servidores y clientes para intercambiar representaciones de datos en la Web en diferentes formatos, sobre todo JSON y XML (de la Fuente, s.f.).

Para el desarrollo de la plataforma se utilizaron como *framework* de desarrollo Django y ReactBootstrap.js. Como lenguajes de programación fueron utilizados Python y JavaScript; para el marcado de hipertexto se utilizó HTML y como lenguaje de hoja de estilos, CSS. En esta plataforma se desea integrar entre otros, servicios como la configuración de DNS, políticas de seguridad, los servidores web (tanto Apache como Nginx).

Para poder integrar un módulo en Nova ARST es necesario que dicho módulo presente una arquitectura como la que presenta la plataforma y además haya sido desarrollado utilizando las mismas tecnologías y lenguajes.

1.4.1 Arquitectura de diseño de la plataforma Nova ARST

La plataforma Nova ARST está diseñada en base a una arquitectura n-capas que interactúan entre ellas a través de interfaces. A continuación, se hace referencia a cuáles son estas capas y que función tiene cada una.

- **Capa de presentación:** esta capa es la encargada de brindar la información al usuario y a su vez también es la capa con la que el usuario interactúa directamente introduciendo los parámetros necesarios para configurar cada uno de los servicios presentes en la plataforma.
- **Capa de aplicación y dominio:** en esta capa es donde se encuentra implementada toda la lógica del negocio. Es la capa encargada de realizar todos los procesos que no son visibles por los usuarios y que este solicita mediante la capa de presentación. También tiene la función de servir de enlace entre la capa de presentación y la de persistencia de datos ya que también se encarga de transmitir los datos introducidos por el usuario a esta última.

- **Capa de acceso y persistencia de datos:** esta capa se encarga de recibir y guardar los datos una base de datos y también de consultarlos cuando estos sean requeridos.

Figura 1 Arquitectura de la plataforma Nova ARST

(Fuente: Elaboración propia)

1.5 Herramientas que administran Nginx

En el presente epígrafe se realiza un análisis de algunas de las herramientas que administran la configuración del servidor web Nginx.

1.5.1 NginxTray

NginxTray es una herramienta privativa, desarrollada para ser una pequeña herramienta que permite manejar fácilmente el servidor web Nginx con un icono de la bandeja. Realiza funciones tales como: iniciar, detener y reiniciar el servidor y en cuanto a la configuración, iniciar y detener el Pre-procesador de Hipertexto

(PHP, del inglés *Hypertext Preprocessor*) FastCGI, definir el directorio PHP FastCGI y establecer el nombre de archivo .exe y la dirección PHP (Molina, 2017).

1.5.2 Módulo de Nginx en Webmin

Según (Molina, 2017) el módulo Nginx en la herramienta Webmin posee funcionalidades que se describen a continuación, ver Figura 1.

- Capacidad de deshabilitar y habilitar *hosts* virtuales: la columna *Status* muestra si el *host* virtual está habilitado o deshabilitado. Para deshabilitar o habilitar un *host* virtual, se puede marcar la casilla debajo de *Select Servers* y usar el menú desplegable *Select action*.
- Mostrar el nombre del fichero: la columna *Server Name* muestra el nombre del fichero que contiene el *host* virtual, al hacer clic se abrirá el *host* virtual para su edición.
- Mostrar la raíz del fichero: la columna *Document Root* muestra el directorio raíz del *host* virtual.
- Permitir una forma sencilla de acceder a la URL del *host* virtual: la columna URL proporciona un hipervínculo para abrir el sitio web en una pestaña.
- Aplicar cambios: luego de actualizar los servidores se debe dar clic en la opción de aplicar cambios (*Apply Changes*).

Se puede utilizar la pestaña *Global Configuration* para modificar el archivo /etc/nginx/nginx.config.

En la pestaña *Create Virtual Host* se pueden crear nuevos *hosts* virtuales.

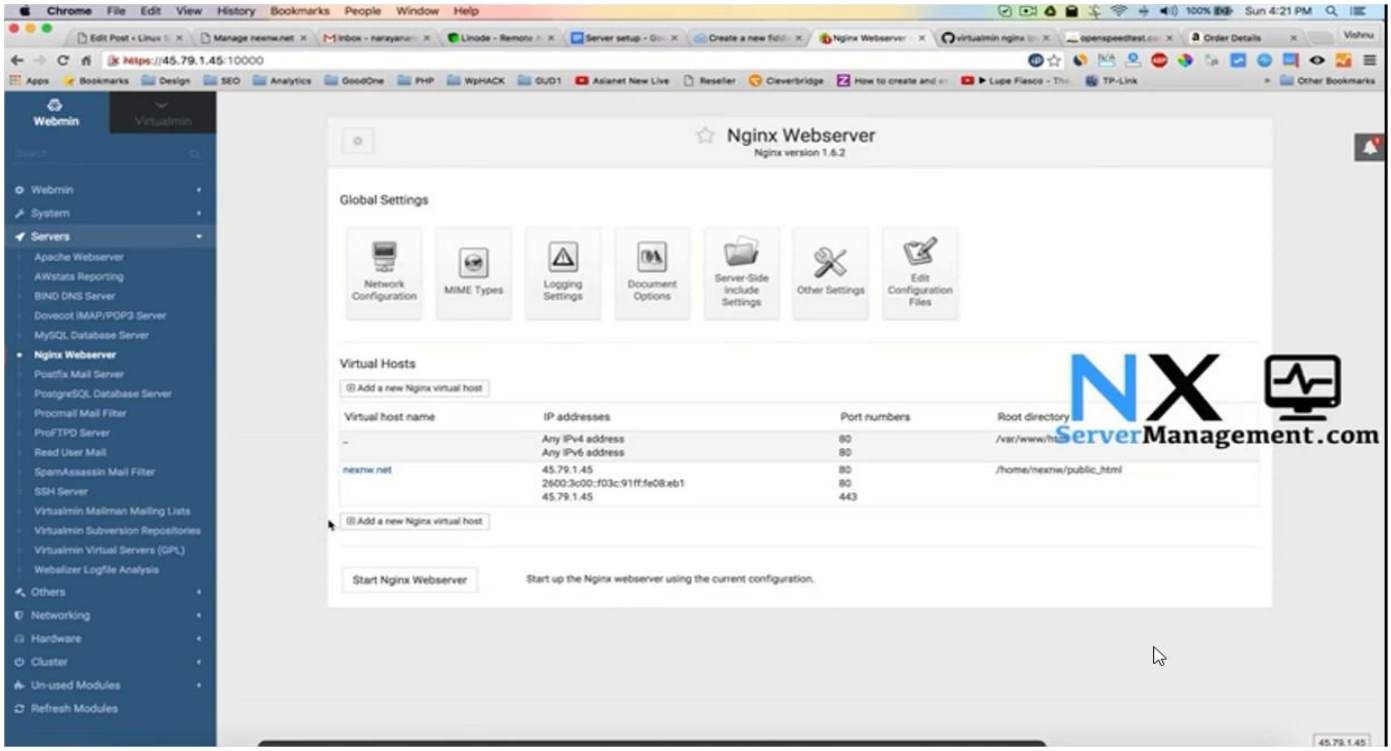


Figura 2 Interfaz del módulo de Nginx en Webmin

1.5.3 Nginx con cPanel

cPanel es uno de los paneles de control de *hosting* más utilizados actualmente para todo tipo de servidores web, pero es su ansiada búsqueda de la estabilidad total, aún no ha implementado soporte nativo para Nginx, ni como servidor web ni como proxy inverso (raiolanetworks, 2014).

- CPNginx: es una extensión para cPanel y WHM de pago que también permite implementar Nginx como servidor web en cPanel, aprovechando las ventajas que ofrece Nginx y además incorpora un módulo para WHMCS.

WHM (del inglés *Web Host Manager*) es un panel de control que brinda al usuario la capacidad de administrar múltiples sitios basados en cPanel.

Resultado de los estudios homólogos

Después del análisis de las herramientas que realizan la administración del servidor web Nginx se llega a la siguiente conclusión:

- NginxTray en cuanto a su interfaz si cumple con los requerimientos, pero es una herramienta de pago por lo que sería más costoso usarla que diseñar otra y además es privativa.
- En cuanto al módulo de Nginx para Webmin, aunque sí es gratuito no es tan sencillo en cuanto su uso ya que la curva de aprendizaje para poder utilizarlo es demasiado extensa.
- Por último, CPNginx a pesar de que posee una interfaz más amigable y es más eficiente, pero es un software de pago además no presenta soporte aun ni como servidor web ni como proxy inverso y como ocurre con NginxTray es privativa.

Tabla 5 Comparación de las herramientas homólogas

(Fuente: Elaboración propia)

Herramienta	Orientado a web	Gratis o de pago	Libre o privativa
NginxTray	No	Pago	Privativa
Módulo de Nginx para Webmin	Sí	Gratis	Libre
CPNginx	Sí	Pago	Privativa

Debido a que ninguna de las herramientas analizadas puede ser integradas en la plataforma Nova ARST ya que no se pudo obtener la información suficiente sobre la arquitectura en que está basado su desarrollo ni las tecnologías que se usaron en el mismo y teniendo en cuenta que justamente esas son las consideraciones para integrar un módulo en Nova ARST se hace necesario la implementación del módulo de Nginx para esta plataforma.

1.6 Metodología de desarrollo de software

En ingeniería de software una metodología de desarrollo constituye un conjunto de métodos, procedimientos, técnicas, herramientas y soportes documentales utilizados para estructurar, planificar y controlar el proceso de desarrollo de sistemas de información (Ramírez, 2018).

Para la correcta implementación de la solución propuesta es necesario la selección de una metodología que guíe el ciclo de vida del proyecto para asegurar un producto de calidad y que cumpla con los estándares de calidad que demandan los usuarios que van a utilizar la plataforma. Se selecciona en consecuencia la

metodología AUP-UCI teniendo en cuenta que es la metodología adaptada al ciclo de vida de los proyectos productivos de la Universidad de las Ciencias Informáticas, es ampliamente usada en el área y es extremadamente flexible al proceso de desarrollo de software.

1.6.1 Metodología de desarrollo de software AUP-UCI

AUP-UCI constituye una variante de AUP (Proceso Unificado Ágil, por sus siglas en inglés) surge con el objetivo de ser una metodología que se adapte al ciclo de vida definido por la actividad productiva en la universidad.

Esta metodología propone tres fases para el desarrollo del software (Inicio, Ejecución y Cierre), siete disciplinas (Modelado de negocio, Requisitos, Análisis y diseño, Implementación, Pruebas internas, Pruebas de liberación, Pruebas de aceptación (Ramírez, 2018).

Inicio: se llevan a cabo las actividades relacionadas con la planeación del proyecto. En esta fase se realiza un estudio inicial de la organización cliente que permite obtener información fundamental acerca del alcance del proyecto, realizar estimaciones de tiempo, esfuerzo y costo y decidir si se ejecuta o no el proyecto.

Ejecución: en esta fase se ejecutan las actividades requeridas para desarrollar el software, incluyendo el ajuste de los planes del proyecto considerando los requisitos y la arquitectura. Durante el desarrollo se modela el negocio, obtienen los requisitos, se elaboran la arquitectura y el diseño, se implementa y se libera el producto.

Cierre: en esta fase se analizan tanto los resultados del proyecto como su ejecución y se realizan las actividades formales de cierre del proyecto

AUP-UCI propone además cuatro escenarios a utilizar para modelar el sistema en los proyectos, de los cuales se escoge:

- **Escenario No 4:** modelar el sistema con HU (Historia de Usuario) cuando no se realice modelado de negocio (Ramírez, 2018).

Esta metodología se adapta al ambiente de trabajo y le permite al cliente acompañar al equipo de desarrollo para convenir los requisitos y así poder implementarlos. De los escenarios descritos anteriormente se selecciona para el modelado del sistema de la solución a desarrollar el escenario número cuatro. Además, se puede concluir que esta metodología se ajusta a cualquier proyecto productivo de la UCI.

1.7 Tecnologías para el desarrollo de la propuesta de solución

El módulo a desarrollar debe ser integrado a la plataforma Nova ARST, por consecuente, las herramientas y tecnologías que se emplean, se deben corresponder con las mismas de la plataforma principal.

1.7.1 Lenguaje para el modelado de la solución

UML 2.0

Lenguaje Unificado de Modelado (UML, del inglés *Unified Modeling Language*) es un lenguaje estándar utilizado para escribir planos de software. Se puede utilizar para visualizar, especificar, construir y documentar los artefactos y las relaciones entre ellos, que se crean durante el desarrollo del software (Molina, 2017). Se emplea para el diseño del módulo, específicamente para la construcción de los diagramas de clases y el diagrama de despliegue.

1.7.2 Herramienta para el modelado de la solución

Una Herramienta de Ingeniería de Software Asistida por Computadora (CASE, del inglés *Computer Aided Software Engineering*) es una herramienta individual para ayudar al desarrollador de software o administrador de proyecto durante una o más fases del desarrollo de software o mantenimiento (Gonzáles & Vera, 2016). En el modelado de la propuesta de solución se emplea Visual Paradigm para UML para generar los artefactos tales como: diagrama de paquetes, diagramas de clases con estereotipos web, diagramas de secuencia, etcétera.

Visual Paradigm 8.0

Es una herramienta CASE diseñada para una amplia gama de usuarios, incluyendo ingenieros de software, analistas de sistemas, analistas de negocios y arquitectos de sistemas, o para cualquier persona que esté interesada en la construcción de forma fiable de los sistemas de software a gran escala con un enfoque orientado a objetos. Ofrece soporte para varias versiones de UML (Gonzáles & Vera, 2016).

1.7.3 Lenguaje de Marcado de Hipertextos

HTML 1.11.2

Es el lenguaje de marcado predominante para la construcción de páginas web. Es usado para describir la estructura y el contenido en forma de texto, así como para complementar el texto con objetos tales como imágenes. HTML (del inglés HyperText Markup Language) se escribe en forma de “etiquetas”, rodeadas por

corchetes angulares (<,>), también puede describir, la apariencia de un documento, y puede incluir un script (por ejemplo, JavaScript), el cual puede afectar el comportamiento de navegadores web y otros procesadores de HTML (Martínez, 2021).

1.7.4 Lenguaje de Hojas de Estilos

CSS 1.10.2

Un lenguaje de Hojas de Estilos en Cascada (CSS, del inglés *Cascading Style Sheets*) es una tecnología que permite crear páginas web de una manera más exacta. Gracias a este lenguaje se pueden hacer muchas más cosas que antes no se podían hacer solamente con HTML, como incluir márgenes, tipos de letra, fondos, colores, definir estilos en un archivo externo a las páginas (Martínez, 2021).

1.7.5 Lenguaje de programación

Un lenguaje de programación es un lenguaje que puede ser utilizado para controlar el comportamiento de una máquina, particularmente una computadora. Consiste en un conjunto de reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos, respectivamente (Saavedra, 2007). En la implementación de la propuesta de solución se emplean los siguientes lenguajes de programación:

Python 3.10.4

Python es un lenguaje de programación de alto nivel que se utiliza para desarrollar aplicaciones de todo tipo. A diferencia de otros lenguajes como Java o .NET, se trata de un lenguaje interpretado, es decir, que no es necesario compilarlo para ejecutar las aplicaciones escritas en Python, sino que se ejecutan directamente por el ordenador utilizando un programa denominado interpretador, por lo que no es necesario “traducirlo” a lenguaje máquina. El Python 3.0 es una versión mayor e incompatible con las anteriores en muchos aspectos, que llega después de un largo período de pruebas el 3 de diciembre de 2008. Muchas de las características introducidas en la versión 3 han sido compatibilizadas en la versión 2.6 para hacer de forma más sencilla la transición entre estas (Banco Santander, 2022).

JavaScript 1.8.5

Es un lenguaje de programación ligero, interpretado, o compilado con funciones de primera clase. Si bien es más conocido como un lenguaje de scripting (secuencias de comandos) para páginas web, es usado en muchos entornos fuera del navegador (MDN web docs, 2022).

1.7.6 Marcos de trabajo

Un marco de trabajo (*framework*) puede verse como una librería o conjunto de librerías donde además de facilitar funciones para su uso, se suele disponer de una sintaxis o metalenguaje específico del *framework* y una forma de organización del código específica (Krall, 2006).

Django 3.1

Django es un *framework* web Python de alto nivel que fomenta un desarrollo rápido y un diseño limpio y pragmático. Creado por desarrolladores experimentados, se ocupa de gran parte de las molestias del desarrollo web, por lo que puede concentrarse en escribir su aplicación. Es gratis y de código abierto. La selección de este *framework* se basa principalmente en la experiencia del equipo de desarrollo en su uso, y el mapeo objeto-relacional (ORM, del inglés *Object Relational Mapping*) de Django abstrae la necesidad de escribir consultas SQL (del inglés *Structured Query Language*) para crear tablas y consultar datos. Es bastante intuitivo de usar y tiene incluidas casi todas las consultas más comunes en su código. Desde filtrados, particionados, uniones e incluso hasta funciones (django, 2022).

React 0.9.0

React es un marco de trabajo de desarrollo que te ayuda a crear interfaces de usuario interactivas de forma sencilla. Se pueden diseñar vistas simples para cada estado en tu aplicación, y se encarga de actualizar y renderizar de manera eficiente los componentes correctos cuando los datos cambien (React, 2022).

1.7.7 Entorno de Desarrollo Integrado

Es un programa compuesto por un conjunto de herramientas para un programador. Puede dedicarse en exclusiva a un solo lenguaje de programación o bien, poder utilizarse para varios. Un IDE (del inglés *Integrated Development Enviroment*) es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica. Los IDE pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes (Martínez, 2021).

Visual Studio Code 1.6.2

Es un editor de código gratuito, ofrece a los desarrolladores soporte integrado para múltiples idiomas. El editor presenta todas las herramientas estándar de un editor de código moderno, incluyendo resaltado de

sintaxis, enlaces de teclado personalizables, coincidencia de corchetes y fragmentos. Control integrado de versiones a través de Git (LARDINOIS, 2015).

1.8 Conclusiones parciales

Los estudios iniciales y la revisión de la bibliografía especializada en el servidor web Nginx, facilitó la comprensión de las características, funcionamiento, configuraciones y administración del mismo permitiendo llegar a las siguientes conclusiones:

- El acercamiento con la plataforma Nova ARST arrojó los requerimientos que debe cumplir el módulo a implementar para integrarlo a dicha plataforma.
- El análisis de las herramientas homólogas permitió evidenciar que las herramientas existentes para administrar Nginx, no solucionan el problema de la presente investigación.
- El estudio de la plataforma ARST precisó que, para la integración del módulo, se debe emplear la metodología de desarrollo de software AUP-UCI y las herramientas: Visual Studio Code y Visual Paradigm. También se determinó usar los lenguajes de programación Python y JavaScript y los *framework*s de desarrollo Django y React.js ya que son las tecnologías que se usaron en el desarrollo de la plataforma.

Capítulo 2: Análisis y diseño del módulo Nginx en la plataforma Nova ARST

El presente capítulo incluye la definición de los requisitos funcionales y no funcionales que debe cumplir el módulo desarrollado a partir de esta investigación. Teniendo en cuenta dichos requisitos funcionales se elaboran las historias de usuarios como plantea la metodología AUP-UCI en su escenario 4. También se describe la arquitectura en que se basa el módulo a desarrollar y los patrones de diseño que serán utilizados.

2.1 Propuesta de solución

A continuación, se presenta la propuesta de solución para darle respuesta al problema de investigación definido previamente y una breve descripción del contexto del problema. Actualmente la configuración de Nginx se realiza a través de la terminal de Nova lo que conlleva a una pérdida de tiempo además de la alta probabilidad de que ocurran errores en la configuración debido a lo alto que es la curva de aprendizaje en el conocimiento de este servicio. Es por ello que en la presente investigación se propone el desarrollo de un módulo que facilite la administración del servidor web Nginx en la plataforma Nova ARST. El módulo permite la administración de dicho servidor web de forma remota desde cualquier computadora que esté conectada al servidor mediante conexiones seguras a través del protocolo SSH (del inglés *Secure Shell*) utilizando una interfaz de usuario en la que cualquier funcionalidad esté al alcance de un *click*, brindando gran comodidad principalmente a usuarios que no tengan un amplio dominio del funcionamiento del servidor web Nginx. El módulo se enfocará en configurar las principales funcionalidades que ofrece Nginx tales como por ejemplo gestionar los virtual *host*. En el próximo epígrafe se definen tanto los requisitos funcionales como los no funcionales y cómo se obtuvieron.

2.2 Levantamiento de requisitos

Según el estándar 1233 de la IEEE: Guía del desarrollo de Especificaciones de Requerimientos de Sistemas, un requisito se define como (Kotonya & Summerville, 1996):

- Condición o capacidad que necesita un usuario para resolver un problema o lograr un objetivo.
- Condición o capacidad que tiene que ser alcanzada o poseída por un sistema o componente de un sistema para satisfacer un contrato, estándar, u otro documento impuesto formalmente.

A partir de lo anteriormente planteado se puede concluir que los requisitos de software son características y funcionalidades que debe cumplir un sistema y están enfocados hacia lo que debe de hacer el software. Además, existen dos tipos de requisitos, los funcionales y los no funcionales. En el siguiente subepígrafe se definen las técnicas que se usaron para la obtención de los requisitos.

2.2.1 Técnicas empleadas en la obtención de requisitos

En el proceso de desarrollo de software siempre se hace necesario realizar un levantamiento de requisitos de forma exitosa, aunque en algunos casos puede ser un proceso complejo pues hay que identificar los requisitos que el sistema debe cumplir en orden de satisfacer las necesidades de los usuarios finales y clientes. Para llevar a cabo esta fase del proceso existen diferentes técnicas, su elección y resultados dependen en gran parte del equipo de desarrollo, como de los propios usuarios o clientes que participen en ellas. A continuación, se muestran las técnicas utilizadas para la identificación de los requisitos (Kotonya & Summerville, 1996):

- **Análisis de sistemas existentes:** Mediante el análisis de sistemas existentes es posible estudiar aplicaciones similares a la que se necesita obtener. Una vez que se tiene la concepción del funcionamiento de un software similar en cuanto a funcionalidades y características es más sencillo identificar los requisitos del sistema que se necesita implementar. Durante la investigación se realiza un estudio de aplicaciones similares a la solución a desarrollar, en las cuales se observan los diseños de sus interfaces, las funcionalidades que brindan y el grado de dificultad en cuanto a su uso.
- **Entrevista:** Se les realiza una entrevista a algunos trabajadores del nodo central de la UCI con el objetivo de saber cuáles son las principales configuraciones que se hacen en el servidor Nginx en la UCI y de qué manera realizan estas configuraciones, si lo hacen mediante alguna herramienta o por la terminal (Ver anexo # 2).

En el siguiente subepígrafe se definen los requisitos funcionales.

2.2.2 Requisitos funcionales

Los requisitos funcionales son capacidades o condiciones que el sistema debe cumplir, cómo debe comportarse en situaciones específicas. En algunos casos también pueden plantear específicamente que no debe hacer el sistema (Somerville, 2005). Los requerimientos funcionales son los que definen las

funciones que el sistema será capaz de realizar, describen las transformaciones que el sistema realiza sobre las entradas para producir salidas (Arias, 2005).

En la siguiente tabla se enumeran los requisitos funcionales que debe tener el módulo a implementar, así como una breve descripción de los mismos y la prioridad que estos precisan.

Tabla 6 Requisitos funcionales

(Fuente: Elaboración propia)

Número	Nombre del requisito	Descripción	Prioridad
RF1	Instalar el servidor web Nginx	Permite realizar la instalación del servidor web Nginx.	Alta
RF2	Desinstalar el servidor web Nginx	Permite realizar la desinstalación del servidor web Nginx.	Alta
RF3	Iniciar el servidor web Nginx	Permite iniciar el servidor web Nginx cuando está detenido.	Alta
RF4	Detener el servidor web Nginx	Permite detener el servidor web Nginx cuando está en funcionamiento.	Alta
RF5	Reiniciar el servidor web Nginx	Permite reiniciar el servidor, es decir, lo detiene y seguidamente lo vuelve a iniciar.	Alta
RF6	Añadir virtual *host* en el servidor web Nginx	Permite añadir un virtual *host* al servidor web Nginx.	Alta
RF7	Modificar los virtual *host* en el servidor web Nginx	Permite modificar un virtual *host* previamente añadido al servidor web Nginx.	Media
RF8	Eliminar virtual *host* del servidor web Nginx	Permite eliminar uno o varios virtual *host* que se encuentren en el servidor web Nginx.	Media

RF9	Habilitar virtual *host*	Permite habilitar un virtual *host* en el servidor web Nginx si se encuentra deshabilitado.	Media
RF10	Deshabilitar virtual *host*	Permite deshabilitar un virtual *host* en el servidor web Nginx si se encuentra habilitado.	Media
RF11	Mostrar tabla de los virtual host	Muestra una tabla en la que aparecen listados todos los virtual host que han sido añadidos al servidor	Media

2.2.3 Requisitos no funcionales

Según los requisitos no funcionales son requerimientos de calidad, que representan restricciones o las cualidades que el sistema debe tener tales como: precisión, usabilidad, seguridad, rendimiento, confiabilidad, performance entre otros. Estos requisitos poseen una naturaleza abstracta e intangible en comparación con los RF y esto hace que sean más difíciles de especificar o documentar formalmente.

Tabla 7 Requisitos no funcionales

(Fuente: Elaboración propia)

No	Requisitos no funcionales
Disponibilidad	
RNF1	Tiempo de actividad: El sistema debe mantenerse en ejecución 24x7x365 con una disponibilidad total de 0.99.
Usabilidad	
RNF2	Garantizar un acceso intuitivo y rápido a los usuarios.
Seguridad	
RNF3	Garantizar que la información sensible solo pueda ser vista por los usuarios con el nivel de acceso autorizado para ello.
Escalabilidad	
RNF4	La aplicación debe poder manejar una carga máxima de 100 usuarios simultáneos.
Software	
RNF5	Los ordenadores donde se ejecutará la aplicación necesitan tener instalado una distribución de GNU\Linux.
RNF6	La aplicación debe desarrollarse con: Python.
Hardware	
RNF7	El equipo donde se despliegue el software debe tener como mínimo, 10 GB disco duro, 256 MB mínimo de RAM.

2.2.4 Historias de Usuario

Las historias de usuario se usan, en el contexto de la ingeniería de requisitos ágil, como una herramienta de comunicación que combina las fortalezas de ambos medios: escrito y verbal. Describen, en una o dos frases, una funcionalidad de software desde el punto de vista del usuario, con el lenguaje que este emplearía. El foco está puesto en qué necesidades o problemas soluciona lo que se va a construir (Menzinsky, y otros, 2022).

Tabla 8 Historia de usuario "Instalar el servidor web Nginx"

(Fuente: Elaboración Propia)

Número: HU – 1	**Nombre del requisito:** Instalar el servidor web Nginx
Programador: Leovaldo Pérez Rojas	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:**3 semanas
Riesgo en Desarrollo: Medio	**Tiempo Real:** no definido
Descripción: El sistema debe permitir al usuario poder instalar el servidor Nginx. • El usuario accede al sistema. • El usuario selecciona la opción instalar haciendo *click* en el botón correspondiente	

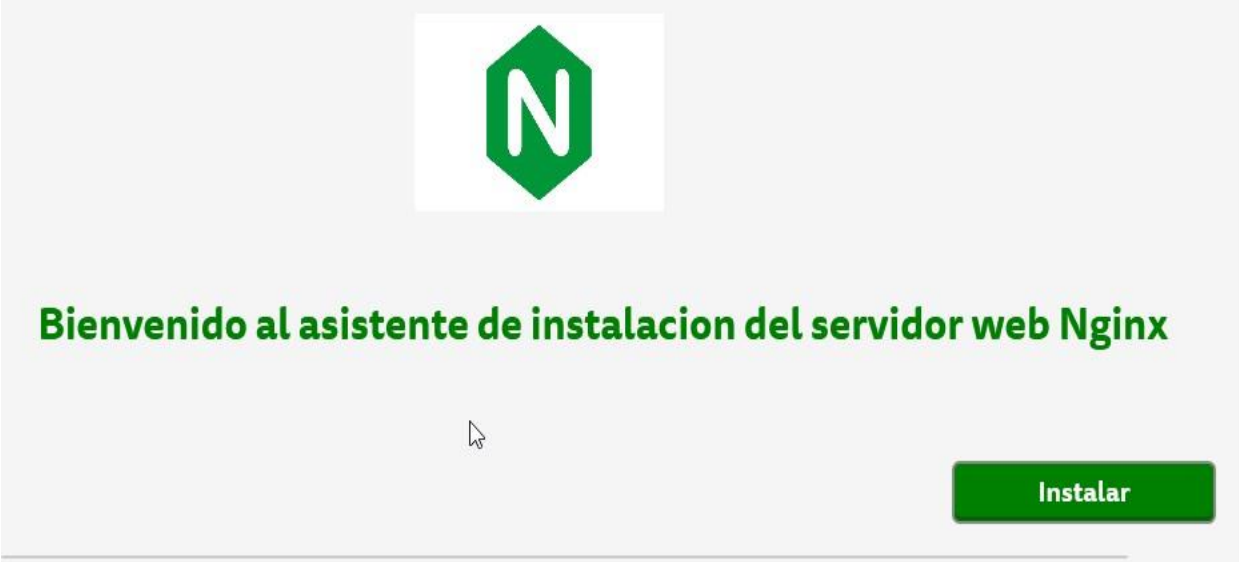


Tabla 9 Historia de usuario "Añadir virtual host en el servidor web Nginx."

(Fuente: Elaboración Propia)

Número: HU – 6	**Nombre del requisito:** Añadir virtual *host* en el servidor web Nginx.
Programador: Leovaldo Pérez Rojas	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:**3 semanas
Riesgo en Desarrollo: Medio	**Tiempo Real:** no definido
Descripción: El sistema debe permitir que el usuario añada un virtual *host* al servidor Nginx. • El usuario accede al sistema. • El usuario selecciona en el menú la pestaña virtual *host*. • El usuario da *click* en el botón Añadir virtual *host*. • El sistema debe mostrar un formulario con los campos necesarios. Los campos a rellenar son:	

- Nombre del fichero: indica el nombre que se le dará al fichero que contiene la configuración del virtual *host*. Es un campo obligatorio y no debe tener más de 255 caracteres.
- Dirección IP: indica la dirección IP del virtual *host* en caso de existir más de una interfaz de red. Se corresponde con el primer argumento de la directiva ***listen***. Es un campo obligatorio. Puede tener varios valores: si la dirección IP es un (*) se asume que se puede acceder a un virtual *host* desde cualquier dirección IP con un puerto determinado. Cuando la directiva ***listen*** aparece solo con el valor del puerto, por defecto se asume la dirección IP del servidor.
- Puerto: se refiere al puerto por el que escucha el servidor asignado al virtual *host*, que debe tener un valor entero positivo entre 1 y 65535. Este responde a la segunda parte del argumento de la directiva ***listen***. Es un campo obligatorio. El puerto puede ser un (*) en caso de que se defina acceder a un virtual *host* mediante una dirección especificada por cualquier puerto.
- Directorio raíz: indica el directorio que contiene lo que será publicado por el virtual *host*. Debe ser la ruta de un directorio que exista en el servidor. Se corresponde con el argumento de la directiva ***root***. Es un campo obligatorio.
- Nombres del servidor: indica la lista de nombres de dominio que se asignan al virtual *host*. Cada elemento debe ser un nombre de dominio válido. Se corresponde con el argumento de la directiva ***server_name***. No es un campo obligatorio.

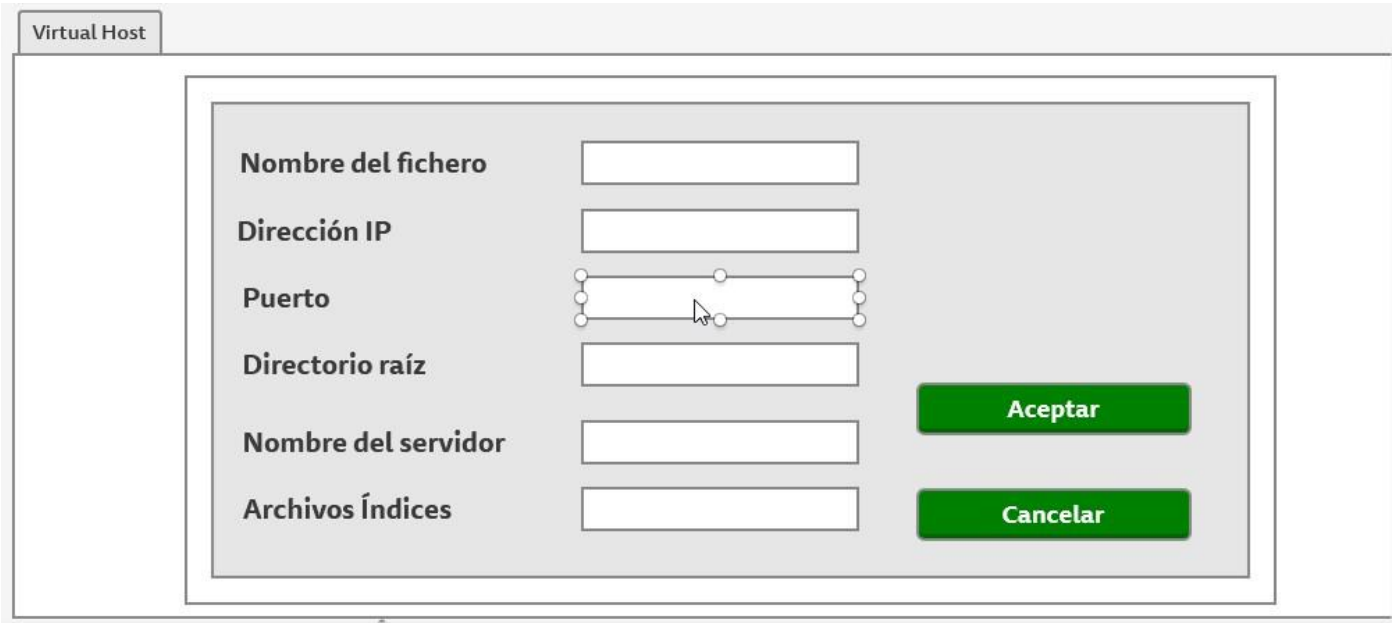
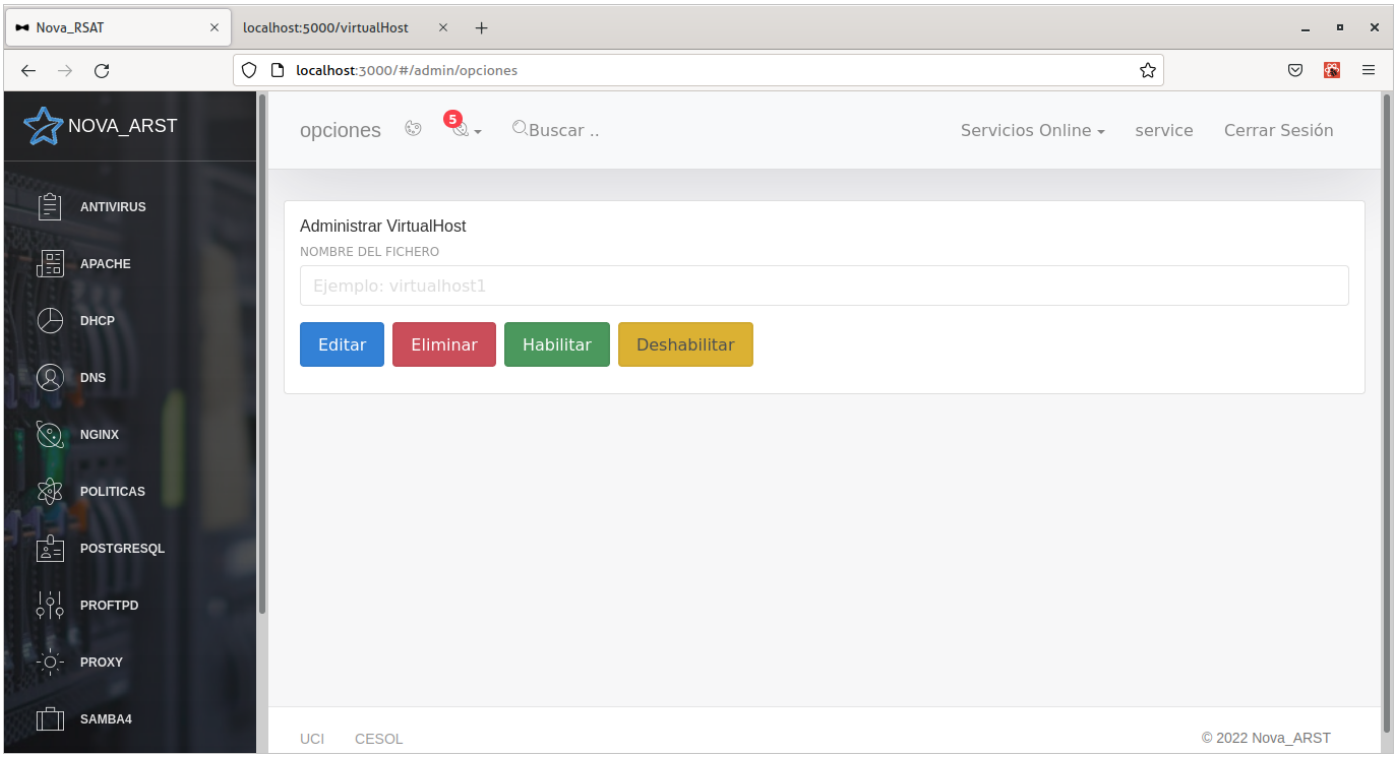


Tabla 10 Historia de usuario "Eliminar virtual host del servidor web Nginx"

(Fuente: Elaboración Propia)

Número: HU – 8	**Nombre del requisito:** Eliminar virtual *host* del servidor web Nginx
Programador: Leovaldo Pérez Rojas	**Iteración Asignada:** 1
Prioridad: Media	**Tiempo Estimado:**3 semanas
Riesgo en Desarrollo: Medio	**Tiempo Real:** no definido
Descripción: El sistema debe permitir al usuario poder eliminar el/los virtual *host* que desee eliminar. • El usuario accede al sistema. • El usuario presiona el botón opciones de la interfaz principal. • El sistema muestra un formulario con el campo nombre del fichero. • El usuario escribe el nombre del fichero y presiona el botón Eliminar.	



2.3 Arquitectura del módulo

Para el diseño del módulo se eligió como arquitectura la n-capas, específicamente en este caso con 3 capas basándose en que es la arquitectura con la que se desarrolló la plataforma Nova ARST y justo la arquitectura es uno de los aspectos a tener a cuenta para poder integrar un módulo en dicha plataforma.

La programación web basada en el modelo de desarrollo por capas es un método que permite programar en capas y usa el concepto de la arquitectura cliente servidor, donde las capas están compuestas por presentación, negocio y acceso a datos.

El módulo cuenta con una capa de presentación que es donde se encuentran todas las plantillas o interfaces con las que el usuario va a interactuar directamente. Una capa intermedia que es en la que se encuentra toda la lógica de la aplicación, es decir todas las clases controladoras que permiten la comunicación entre las otras dos capas, y la última capa, la capa de acceso a datos que es donde se guardan los datos nuevos o se pueden consultar datos que se encuentren anteriormente introducidos. Esta última capa está integrada por las clases encargadas de manipular todos los ficheros de configuración del servidor Nginx.

Figura 3 Arquitectura del módulo

(Fuente: Elaboración propia)

2.3.1 Diagrama de paquetes para representar la arquitectura

Un diagrama de paquete es un diagrama de bloques que tiene paquetes cuyo propósito es una organización modelo (Douglass, 2016). Los diagramas de paquetes son diagramas estructurales que se emplean para mostrar la organización y disposición de diversos elementos de un modelo en forma de paquetes. Un paquete es una agrupación de elementos UML relacionados, como diagramas, documentos, clases o, incluso, otros paquetes. Cada elemento está anidado dentro de un paquete, que se representa como una carpeta de archivos dentro del diagrama, y que luego se organiza jerárquicamente dentro del diagrama.

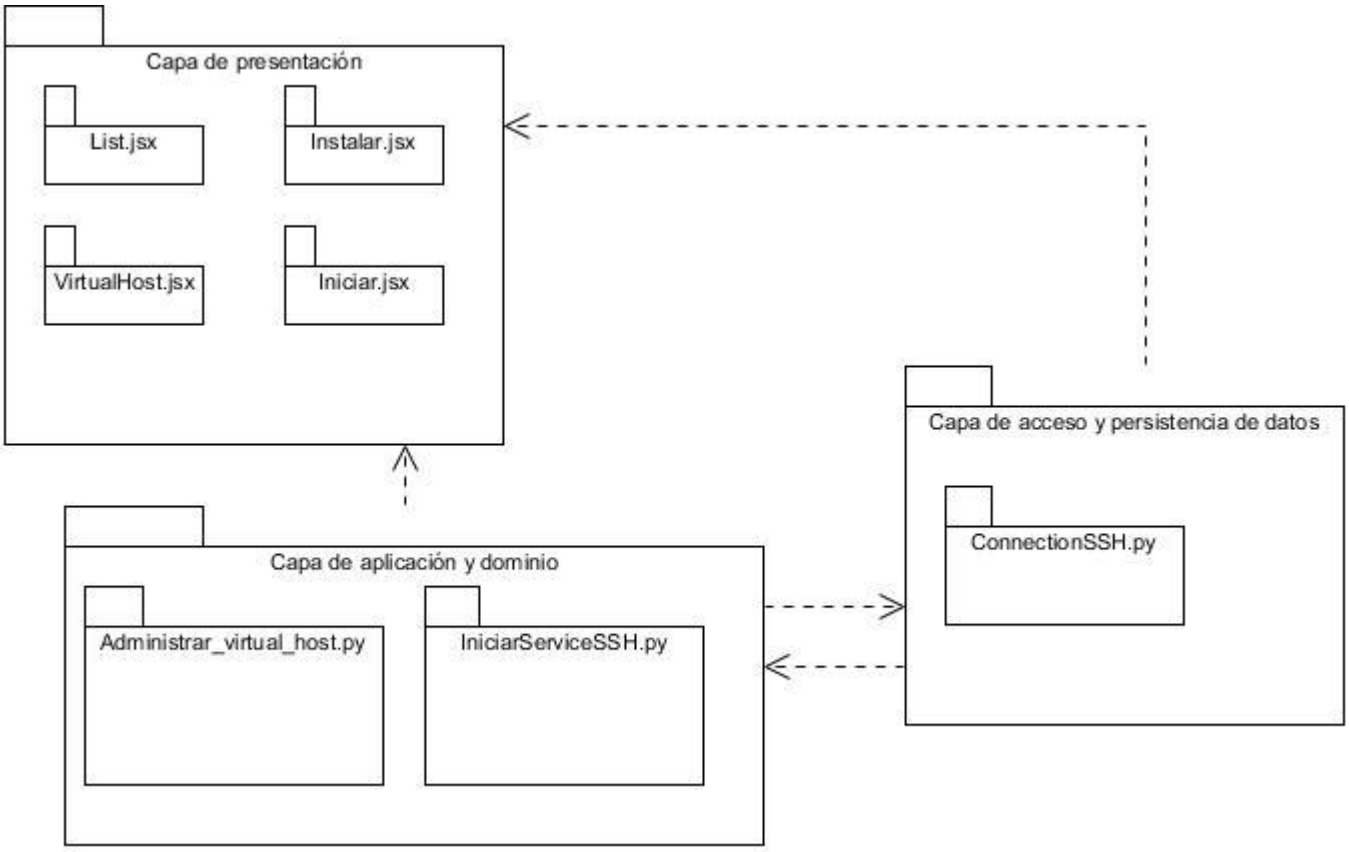


Figura 4 Diagrama de paquetes para representar la arquitectura

(Fuente: Elaboración propia)

2.3.2 Diagrama de clases con estereotipos web

El diagrama de clases es un gráfico que representa el comportamiento del sistema en forma gráfica y es parte del diseño de software. Debe desarrollarse antes de la generación y, a su vez, contar con especificación formal para aplicaciones web, tal como el modelo de desarrollo por capas (Fajardo, 2019). A continuación, se muestran los diagramas de clases correspondientes.

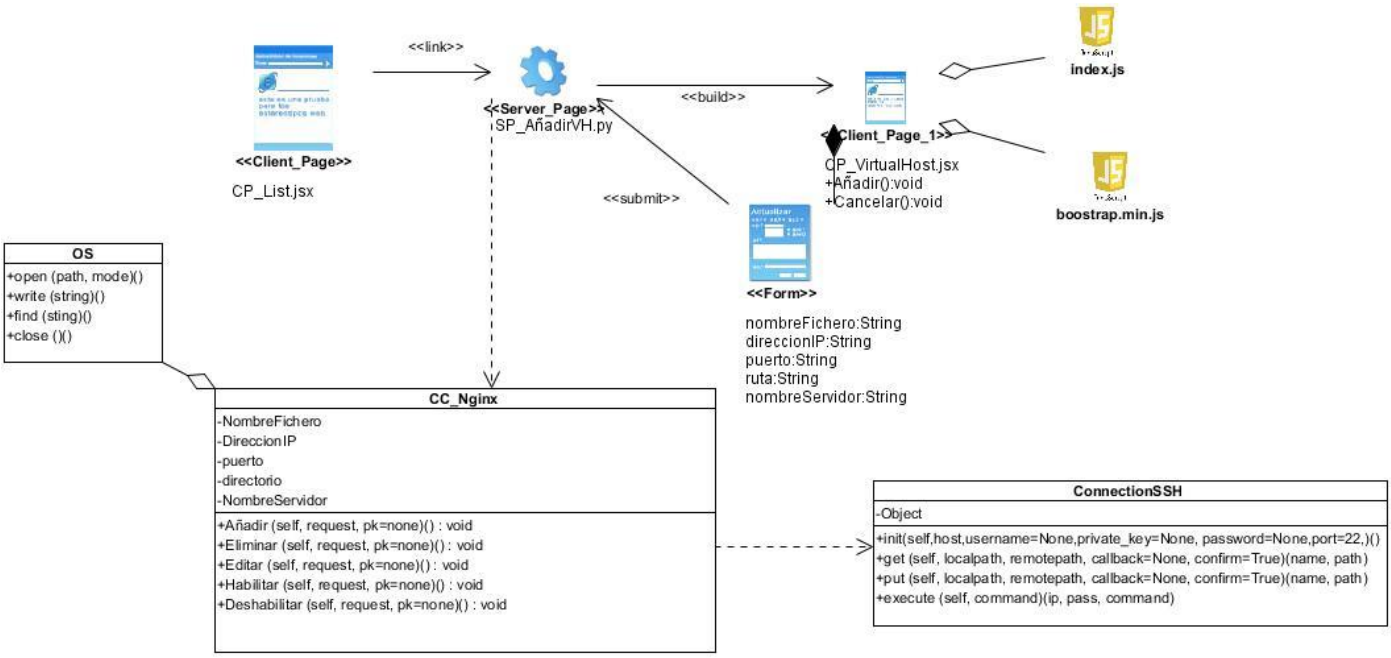


Figura 5 Diagrama de clases: Añadir virtual host

(Fuente: Elaboración propia)

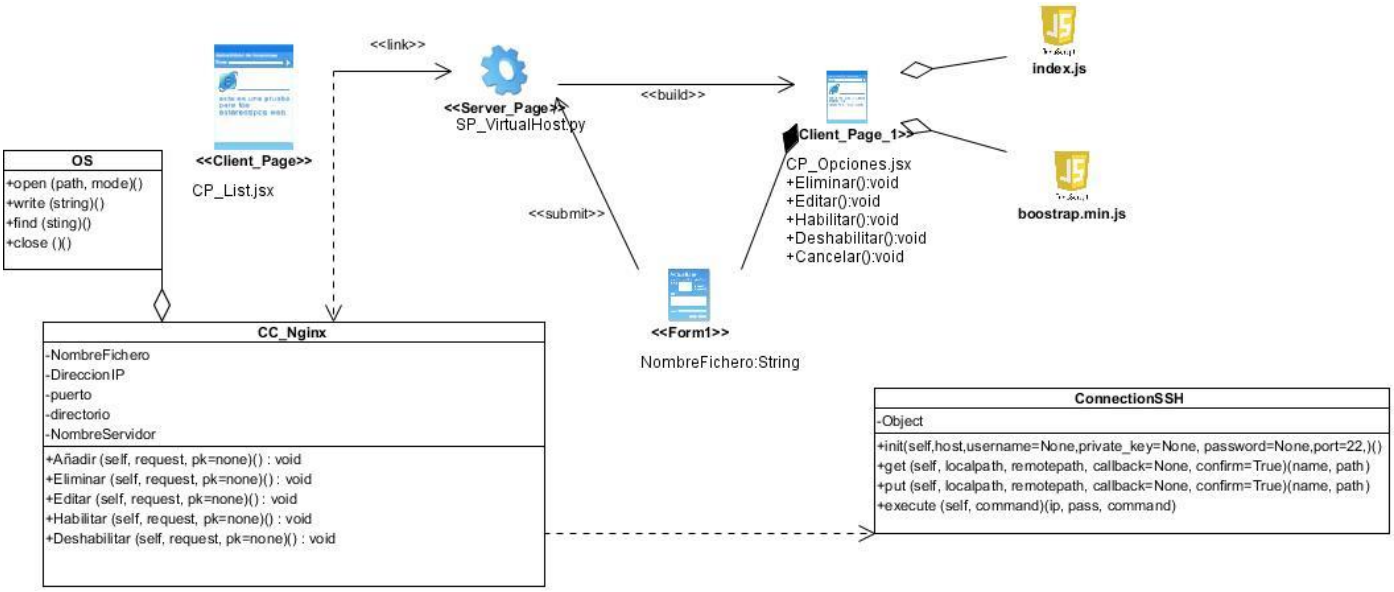


Figura 6 Diagrama de clases: Eliminar virtual host

(Fuente: Elaboración propia)

2.4 Patrones de diseños

Los patrones de diseño de software son el esqueleto de las soluciones a problemas comunes en el desarrollo de software. Brindan una solución a problemas de desarrollo de software que están sujetos a contextos similares (Gutiérrez & Gato, 2018). En esta investigación se utilizan los patrones GRASP (*General Responsibility Assignment Software Patterns),* más conocidos como patrones de asignación de responsabilidades y los patrones GOF (*The Gang of Four*).

2.4.1 Patrones GRASP

Los patrones GRASP describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones (LARMAN, 1999). En el desarrollo del módulo se utilizaron los siguientes patrones GRASP:

- **Patrón Experto:** Se basa en asignar una responsabilidad a la clase que cuenta con la información necesaria para cumplir dicha responsabilidad. Permite conservar el encapsulamiento, ya que los objetos se valen de su propia información para realizar lo que se le oriente (Gutiérrez & Gato, 2018).

Este patrón se puede evidenciar claramente en la clase ***ConectionSSH***, que es la única encargada de comunicar al ordenador donde se encuentre instalada la plataforma Nova ARST con el que actúa como servidor donde será administrado el servidor web Nginx.

```
class ConnectionSSH(object):
    """Connects and logs into the specified hostname.
    Arguments that are not given are guessed from the environment."""

    def __init__(self,
                 host,
                 username=None,
                 private_key=None,
                 password=None,
                 port=22,
                 ):
```

Figura 7 Definición de la clase ConectionSSH

(Fuente: Elaboración propia)

- **Patrón Creador:** El patrón Creador guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. El objetivo fundamental de este patrón es encontrar un creador que se conecte con el objeto producido en cualquier evento. Al comportarse como creador se da soporte al bajo acoplamiento (Gutiérrez & Gato, 2018). En el módulo desarrollado se evidencia este patrón en la clase ***Nginx*** ya que en esta se crean instancias de la clase ***ConectionSSH*** para ejecutar los comandos que se necesitan en el servidor y copiar ficheros entre la el servidor y el cliente.

```
@action(methods=['post'],detail=False,url_path='Eliminar',url_name='Eliminar')
def Eliminar (self,request,pk=None):
    obtenido = request.data
    nombre = obtenido['nombre']
    cnx = ConnectionSSH(str(IP_SERVER),USER_SERVER, password = PASSWORD_SERVER)
    outputComand = cnx.execute(f'echo "{PASSWORD_SERVER}" | sudo -S rm -r /etc/nginx/sites-available/{nombre}.conf')
    return Response("asd")
```

Figura 8 Creación de un objeto de la clase ConectionSSH

(Fuente: Elaboración propia)

- **Patrón Bajo Acoplamiento:** Este patrón asigna una responsabilidad para mantener el bajo acoplamiento. La idea es tratar de que una clase no dependa de muchas otras, así esa clase no tendrá muchas dependencias, lo que facilitará la reutilización de la misma y se reducirá el impacto de los cambios (Gutiérrez & Gato, 2018). La clase con mayor número de relaciones es la ***Nginx*** que es a su vez solo se relaciona con 3 clases (services, os, ConectionSSH), por lo que se garantiza que hay un bajo acoplamiento entre clases.
- **Patrón Alta Cohesión:** Una clase con alta cohesión tiene un número relativamente pequeño de métodos, con funcionalidades altamente relacionadas, y no realiza mucho trabajo, colaborando con otros objetos para compartir el esfuerzo si la tarea es extensa. Las clases con alta cohesión son relativamente fáciles de mantener, entender y reutilizar (Gutiérrez & Gato, 2018). Un ejemplo donde se evidencia este patrón es en la clase ***ConectionSSH*** ya que esta solo contiene los métodos necesarios para garantizar la conexión con el servidor.

```
def get(self, remotepath, localpath=None):
    """Copies a file between the remote host and the local host."""
    if not localpath:
        localpath = os.path.split(remotepath)[1]
    self._sftp_connect()
    self._sftp.get(remotepath, localpath)

def put(self, localpath, remotepath, callback=None, confirm=True):
    """Copies a file between the local host and the remote host."""
    if not remotepath:
        remotepath = os.path.split(localpath)[1]
    self._sftp_connect()
    self._sftp.put(localpath, remotepath)

def execute(self, command):
    """Execute the given commands on a remote machine."""
    channel = self._transport.open_session()
    channel.exec_command(command)
    output = channel.makefile('rb', -1).readlines()
    if output:
        return output
    else:
        return channel.makefile_stderr('rb', -1).readlines()
```

Figura 9 Métodos de la clase ConectionSSH

(Fuente: Elaboración propia)

- **Patrón Controlador:** Este patrón se utiliza para manejar y controlar los eventos del sistema. Con la utilización de este patrón se logra separar la lógica de negocio de la capa de presentación. De esta manera se logra mayor control sobre el sistema y se favorece la reutilización de código (Gutiérrez & Gato, 2018). Una clase donde se evidencia el uso de este patrón en el módulo es la ***Nginx***, que es la en la que están implementadas todas las funcionalidades que manipulan al servidor web Nginx.

2.4.2 Patrones GOF

Los patrones GOF complementan a los patrones GRASP y en ocasiones se puede encontrar una contraposición entre este tipo de patrones, e incluso, podría inferirse que algunos patrones GOF son variantes de los patrones GRASP, es por ello que la decisión de utilizar uno u otro debe tomarse con precaución y aplicarse solo en el ámbito necesario (Gutiérrez & Gato, 2018).

En el desarrollo del módulo se emplearon los siguientes patrones GOF dependiendo de los *framework* que se utilizaron para su desarrollo:

- **Decorator (Decorador):** es un patrón de diseño estructural que permite añadir funcionalidades a objetos colocando estos objetos dentro de objetos encapsuladores especiales que contienen estas funcionalidades. Este patrón es muy utilizado en el lenguaje de programación **python,** concretamente es muy extendido en los *frameworks* web en funciones variadas relacionadas con la lógica del negocio y la lógica de la aplicación (Guerrero, Suárez, & Gutiérrez, 2013). El empleo de este patrón se evidencia al utilizar el decorador *@action*.

```
@action(methods=['post'],detail=False,url_path='Eliminar',url_name='Eliminar')
```

Figura 10 Patrón Decorator

(Fuente: Elaboración propia)

2.5 Conclusiones Parciales

A partir del desarrollo del presente capitulo se llegó a las conclusiones siguientes:

- La captura de los requisitos y la elaboración de las historias de usuario correspondientes permitieron comprender mejor las funcionalidades de la aplicación que se desea desarrollar y el comportamiento de la misma.
- La utilización de la arquitectura 3-capas y los patrones de diseño contribuyeron al diseño de la aplicación, proporcionando una estructura para la misma y posibilitando el empleo de buenas prácticas de programación y la reutilización de código.
- Como resultado principal de este capítulo quedó diseñada la propuesta de solución para la problemática planteada, detallada a través de los requisitos funcionales y no funcionales, las historias de usuario, la arquitectura y el uso de patrones de diseño.

Capítulo 3: Implementación y validación del módulo para la administración remota del servidor web Nginx desde la plataforma Nova ARST.

La disciplina de implementación en el proceso de desarrollo de software, es el mecanismo mediante el cual se ponen en práctica y se vuelven tangibles todas las descripciones y modelaciones propuestas en la fase de análisis y diseño, es el complemento del trabajo de las fases que la preceden. Para un despliegue exitoso del módulo desarrollado primero debe pasar por un conjunto de pruebas que validen el correcto funcionamiento del mismo, así como el cumplimiento de todos los requisito funcionales y no funcionales definidos en la fase del diseño. En el presente capítulo se exponen las especificaciones asociadas a la implementación del módulo, se describen los estándares de codificación empleados, y también el diseño de los casos de pruebas y el resultado que estos arrojaron en cada iteración.

3.1 Implementación del sistema.

Luego de terminada la fase de diseño se da paso a la implementación del módulo, logrando concebir la arquitectura y el sistema como un todo.

3.1.1 Diagrama de componentes

Los diagramas de componentes representan las relaciones entre los componentes individuales del sistema mediante una vista de diseño estática. Pueden ilustrar aspectos de modelado lógico y físico. En el contexto del UML, los componentes son partes modulares de un sistema independientes entre sí, que pueden remplazarse con componentes equivalentes. Son auto contenidos y encapsulan estructuras de cualquier grado de complejidad. Los elementos encapsulados solo se comunican con otros a través de interfaces. Los componentes no solo pueden proporcionar sus propias interfaces, sino que también pueden utilizar las interfaces de otros componentes, por ejemplo, para acceder a sus funciones y servicios. A su vez, las interfaces de un diagrama de componentes documentan las relaciones y dependencias en una arquitectura de software (IONOS, 2020).

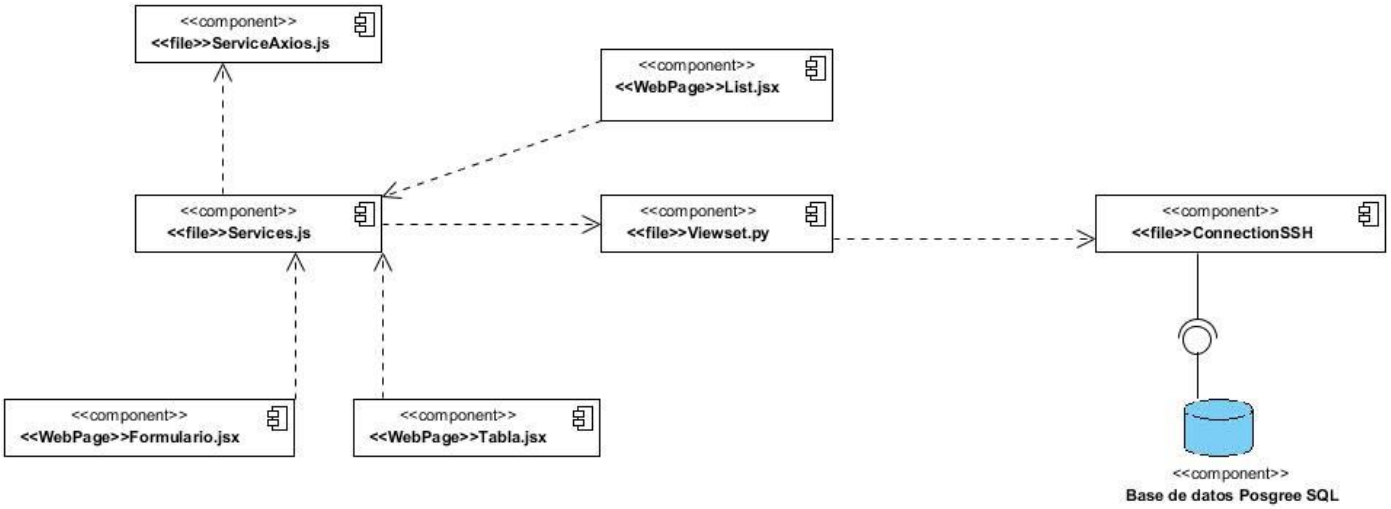


Figura 11 Diagrama de componente. Crear virtual host

(Fuente: Elaboración propia)

Descripción de los componentes utilizados:

Dentro de la capa de presentación se encuentran los componentes:

List.jsx: página principal del módulo desde la cual se accede a todas las funcionalidades.

Services.js: componente que contiene las llamadas a las funcionalidades que se ejecutan en el *back-end*.

ServiceAxios.js: componente que contiene los métodos **get**, **post**, **put**, **del** para comunicar el *front-end* con el *back-end*

Formulario.jsx: componente contenedor del formulario que se utiliza para crear un *Virtual Host*

Tabla.jsx: componente que contiene la tabla donde se muestra la lista de los *Virtual Host* que se han creado.

Dentro de la capa de aplicación y dominio se encuentran los componentes:

ViewSet.py: componente en el cual se realizan las acciones necesarias para administrar cada *Virtual Host*.

Dentro de la capa de acceso y persistencia de datos se encuentran los componentes:

ConnectionSSH.py: En este componente están implementados los métodos que se encargan de ejecutar un comando en el servidor, y copiar un archivo entre el cliente y el servidor

NuevoVirtualHost: representa el fichero que corresponde a cada *Virtual Host* nuevo que se crea.

3.1.2 Estándares de codificación

Un estándar de codificación completo comprende todos los aspectos de la generación de código. Si bien los programadores deben implementar un estándar de forma prudente, este debe tender siempre a lo práctico. Un código completo debe reflejar un estilo armonioso, como si un único programador hubiera escrito todo el código de una sola vez (Arevalo, 2012).

Para facilitar el entendimiento del código y fijar un modelo a seguir, se establecen estándares de codificación. La mantenibilidad del código es la facilidad con que el sistema de software puede modificarse para añadirle nuevas características, modificar las ya existentes, depurar errores, o mejorar el rendimiento.

Estándares para JavaScript y JSX

Reglas a seguir para el trabajo con las clases del módulo:

- Los nombres de clases deben ser sustantivos y deben tener la primera letra en mayúsculas. Si el nombre es compuesto, cada palabra componente deberá comenzar con mayúsculas. Los nombres serán simples y descriptivos. Debe evitarse el uso de acrónimos o abreviaturas, salvo en aquellos casos en los que dicha abreviatura sea más utilizada que la palabra que representa (URL, HTTP, entre otras.).
- Se utilizan los comentarios en caso de ser necesario para explicar la utilidad de la clase.

```
//Ejecuta la funcion que se encuentre en la ruta /nginx/formulario en el backend
servicesAxios.post("/nginx/formulario/",form);
```

Figura 12 Estándar para comentarios

(Fuente: Elaboración propia)

- En **React**, ***className*** remplaza el atributo ***class*** de **html** y ***htmlFor*** remplaza el atributo ***for***.

```
<Card className={styles.cardCss}>
```

Figura 13 Uso del className

(Fuente: Elaboración propia)

```
<label htmlFor="ip">Direccion IP</label>
```

Figura 14 Uso del htmlFor

(Fuente: Elaboración propia)

- Cuando se utilizan estilos fuera de línea, el nombre del atributo debe tener preferentemente la palabra clave *Style*.

```
{styles.vistaInstalar}>
```

Figura 15 Uso del style

(Fuente: Elaboración propia)

- Cuando hay muchos atributos en una sola etiqueta, se escribe un atributo por línea y la etiqueta de cierre en la última línea y separada.

```
<input
    type = "text"
    name = "ip"
    placeholder = "Ejemplo: 10.0.0.1"
    id = "ip"
    onChange = {hadleChange}
    value = {form.ip}
/>
```

Figura 16 Atributos en una sola etiqueta

(Fuente: Elaboración propia)

Reglas a seguir para nombrar variables y métodos:

- Los nombres deben ser descriptivos y concisos. No usar grandes frases y solo abreviaturas pequeñas.

```
CreateVH(form);
```

Figura 17 Declaración de métodos

(Fuente: Elaboración propia)

- El identificador para las variables y los parámetros serán con letras en minúsculas y en caso de ser un nombre compuesto se divide cada palabra con mayúscula inicial.
- Los métodos deben ser verbos escritos en minúsculas. Cuando el método esté compuesto por varias palabras cada una de ellas tendrá la primera letra en mayúsculas.

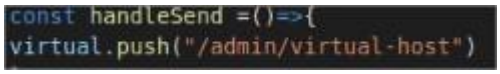


Figura 18 Identificador para métodos

(Fuente: Elaboración propia)

Estándares para python

- **Operadores:** Los operadores binarios, que se utilizan entre dos valores, deben separarse de estos valores, a ambos lados del operador, por un espacio. Por ejemplo, numero = 5, en lugar de numero=5. Esto se aplica a operadores como +, -, *, /, =, ==, >, <. Los operadores unarios (++, -) no deben tener separación. Por ejemplo, numero++.

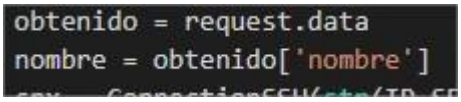


Figura 19 Operadores

(Fuente: Elaboración propia)

- **Nombres de archivos:** Los nombres de archivos deben escribirse siempre en minúsculas. La única excepción son los archivos de documentación, que tendrá extensión **.txt**.

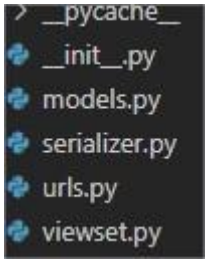


Figura 20 Nombres de archivos

(Fuente: Elaboración propia)

- **Importación:** Las importaciones siempre se colocan al comienzo del archivo, simplemente luego de cualquier comentario o documentación del módulo, y antes de globales y constantes.

```
from rest_framework.viewsets import ModelViewSet
from django.conf import settings
from rest_framework import status
from rest_framework.response import Response
from rest_framework.decorators import action
from .serializer import NginxZerializer
from .models import Nginx, VirtualHost
from django_filters.rest_framework import DjangoFilterBackend
import os
```

Figura 21 Importación

(Fuente: Elaboración propia)

3.2 Diagrama de despliegue

El modelo de despliegue consiste en un modelo de objetos que tiene como objetivo describir la distribución física del sistema en términos de cómo se distribuye la funcionalidad entre las estaciones de trabajo (denominados nodos). La vista de despliegue representa la disposición de las instancias de componentes de ejecución en instancias de nodos conectados por enlaces de comunicación. Los nodos representan objetos físicos existentes en tiempo de ejecución, sirven para modelar recursos que tiene memoria y capacidad de proceso y puede ser tanto ordenadores como dispositivos, memoria o personas (Sommerville, 2011). A continuación, se muestra el diagrama de despliegue para la solución propuesta.

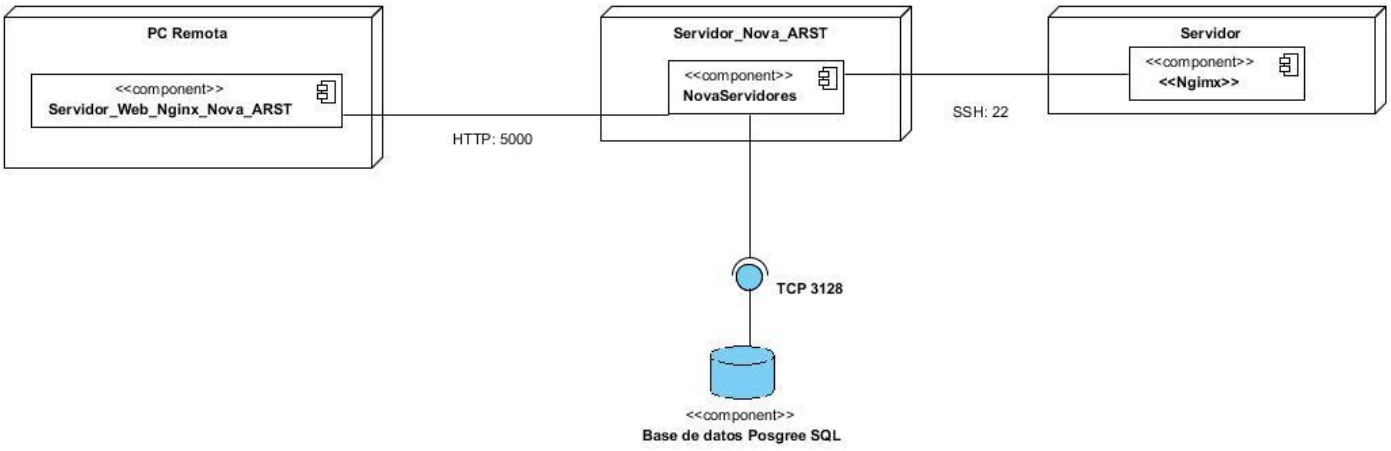


Figura 22 Diagrama de despliegue

(Fuente: Elaboración propia)

A continuación, se muestra una descripción del diagrama de despliegue propuesto para el módulo.

PC Remota: Esta computadora tiene la capacidad de acceder a un ordenador o dispositivo desde cualquier ubicación remota. Esta PC de escritorio contiene el componente (Configuración de archivos para Nova ARST) el cual es la aplicación de escritorio que permite realizar las configuraciones del del servidor web Nginx de manera remota utilizando SSH.

SSH: es un protocolo que trabaja como el parámetro de seguridad que procura la administración de los aspectos relacionados a la conexión entre redes**.** Es bien conocido ya que a través de los servidores se realiza una constante transmisión de información desde un computador a otro. Desde SSH se pueden realizar copias seguras de los archivos que se transmiten a través de la dirección seleccionada, en caso de necesitarlas. Por defecto SSH utiliza el protocolo TCP de la capa de transporte, y el número de puerto es el 22.

Nova ARST: Consiste en una herramienta que permite la configuración remota del servidor.

Nova Servidores: Distribución de GNU/Linux orientada a servidores usando estándares abiertos y adaptada a los entornos de las empresas cubanas. Entre sus características se encuentran la configuración fácil e intuitiva a través de la herramienta nova-manager, destinada a la administración de los servicios telemáticos y la compatibilidad con hardware obsoleto en el entorno empresarial.

Base de datos Posgree SQL: Es la base de datos de la plataforma Nova ARST, donde se guardan datos como los usuarios que tienen acceso a la misma.

Servidor: Servidor en el que será instalado y configurado el servidor web Nginx.

HTTP: Protocolo de transporte de hipertexto.

3.3 Evaluación de la propuesta de solución.

Durante la etapa de implementación, pueden cometerse algunos errores y pueden pasarse por alto algunos elementos que son importantes para el correcto funcionamiento del sistema. Por tal motivo, es esencial llevar a cabo la fase de validación, en la cual, a través de varios tipos y métodos de pruebas de software (estrategia de pruebas), se pretende comprobar el cumplimiento de las especificaciones del diseño y de la

codificación, identificar los posibles errores cometidos y validar la solución propuesta en los capítulos anteriores (Labrada & Aragón, 2013).

El proceso de evaluación del módulo para la administración del servidor web Nginx en la plataforma Nova ARST se realiza mediante la estrategia de pruebas. La estrategia de pruebas de software proporciona un mapa que describe los pasos que se darán para realizarlas, indica cuándo se planea y se darán dichos pasos, además cuánto tiempo, esfuerzo y recursos consumirán. Un software se prueba para descubrir los errores cometidos, si se realiza sin ningún plan seguramente se desperdicia tiempo y esfuerzo.

3.3.1 Pruebas de integración

Abordan los conflictos asociados con los problemas duales de verificación y construcción de programas. Durante la integración, se usan más las técnicas de diseño de casos de prueba que se enfocan en entradas y salidas (Pressman R. , 2002).

Al realizar la prueba de integración se examinaron las interfaces entre grupos de componentes o subsistemas para asegurar que son llamados cuando es necesario y que los datos o mensajes que se transmiten son los requeridos. Con esta prueba se demuestra si la comunicación entre los distintos componentes presentes en el software es funcional. Durante esta se ensamblan los módulos independientes, se da forma al software al completo y se verifica el proceso a conciencia.

Una de las ventajas de utilizar esta prueba se encuentra en la oportunidad de llevar a cabo pruebas de una manera paralela, lo que aporta flexibilidad extra en el proceso de calendarización. Para ello se opta por la elección de un framework en el que las pruebas se puedan combinar con el desarrollo y con la supervisión simplificada de los procesos, especialmente en aquellos casos en los que las pruebas puedan ser un poco más complejas.

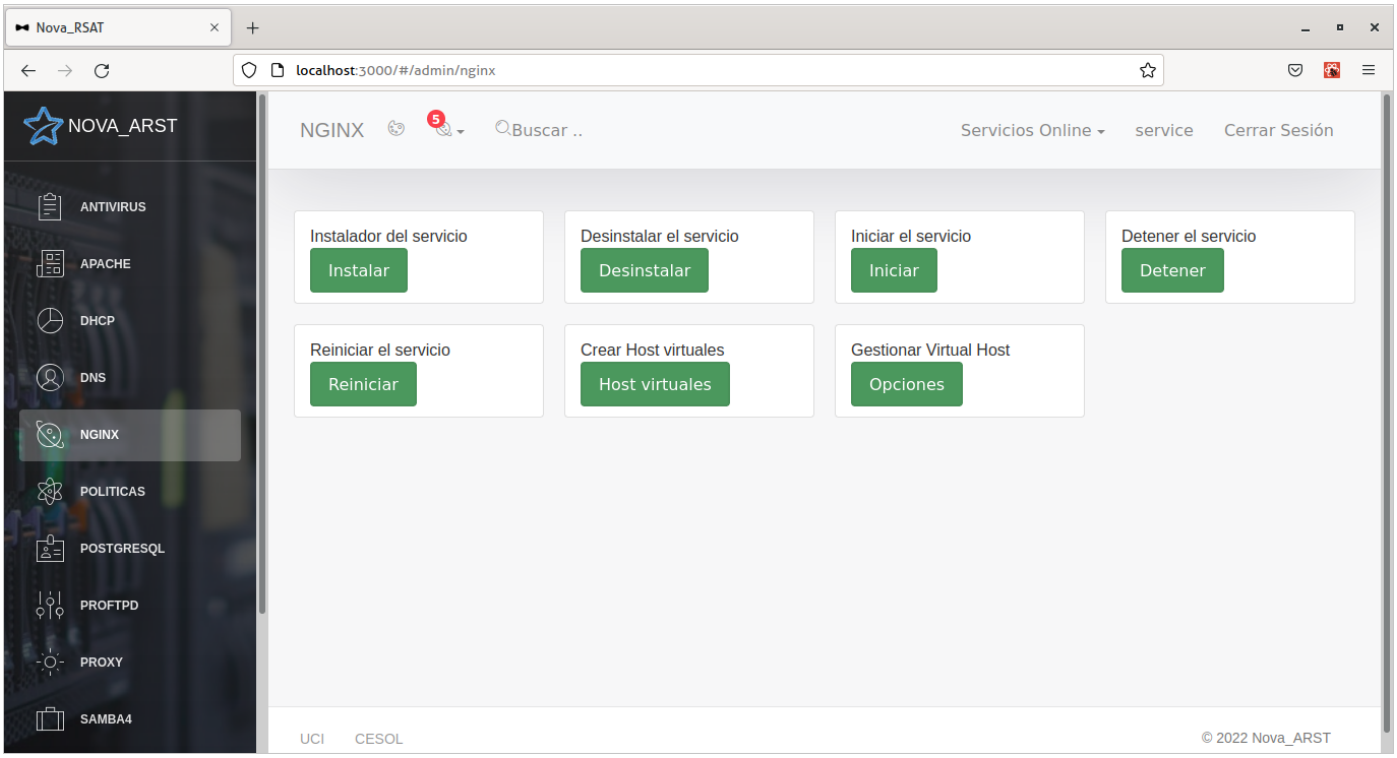


Figura 23 Módulo integrado en la plataforma Nova ARST

(Fuente: Elaboración propia)

3.3.2 Pruebas funcionales

Se realizaron pruebas funcionales las cuales llevan a cabo para comprobar las características críticas para el negocio, la funcionalidad y la usabilidad. Estas garantizan que las características y funcionalidades del software se comportan según lo esperado sin ningún problema (Pressman R. , 2002).

Las **Pruebas de Caja Negra**, es una técnica de pruebas de software en la cual la funcionalidad se verifica sin tomar en cuenta la estructura interna de código, detalles de implementación o escenarios de ejecución internos en el software (Pressman R. , 2002).

Técnica de partición de equivalencias

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 (Pressman R. S., 2005).

El diseño de los casos de prueba para la partición se basa en una evaluación de clases de equivalencia para la condición de entrada. Con el propósito de realizar estas pruebas al método para la administración remota del servidor web Nginx, se diseñó un conjunto de casos de pruebas a los requisitos funcionales identificados en el capítulo anterior. A continuación, se muestran uno de los casos de prueba:

Tabla 11 Pruebas de caja negra

(Elaboración propia)

Caso de prueba 6: RF7 Añadir virtual host en el servidor web Nginx				
Escenario	**Descripción**	**Nombre**	**Respuesta del sistema**	**Flujo central**
1. Se crea un nuevo virtual host con los datos recogidos en el formulario.	El usuario presiona el botón de enviar dejando campos vacíos.	virtualhost1.conf	El sistema muestra una alerta de que no se pueden dejar campos vacíos.	1.1 El usuario selecciona la opción Crear virtual host de la interfaz principal. 1.2 El sistema muestra el formulario correspondiente en el cual el usuario debe digitar los datos. 1.3 El usuario presiona el botón enviar sin rellenar ningún campo del formulario. 1.4 El sistema muestra una alerta de que no se pueden dejar campos vacíos

2. Se crea un nuevo virtual host con los datos recogidos en el formulario.	El usuario introduce correctamente todos los datos en el formulario.	Virtualhost2.conf	El sistema muestra el nuevo virtual host creado en la tabla donde se listan los mismos.	2.1 El usuario selecciona la opción Crear virtual host de la interfaz principal. 2.2 El sistema muestra el formulario correspondiente en el cual el usuario debe digitar los datos. 2.3 El usuario digita todos los datos correctamente y presiona el botón enviar. 2.4 El sistema muestra el nuevo virtual host creado en la tabla.
3. Se crea un virtual host con una dirección IP q no es valida	El usuario introduce la dirección IP: 10.257.4	V Virtualhost3.conf	El sistema muestra una notificación de que la dirección IP no es válida.	3.1 El usuario selecciona la opción Crear virtual host de la interfaz principal. 3.2 El sistema muestra el formulario correspondiente en el cual el usuario debe digitar los datos. 3.3 El usuario digita todos los datos correctamente excepto la dirección IP. 3.4 El sistema muestra una notificación de que la dirección IP no es válida.

Caso de prueba 8: RF8 Eliminar virtual *host* del servidor web Nginx				
5. Se elimina un virtual host creado.	El usuario introduce un nombre de virtual host que no existe.	Virtualhost4.conf	El sistema muestra una notificación de que no existe ningún virtual host con ese nombre.	5.1 El usuario presiona el botón opciones del menú principal. 5.2 El sistema muestra un formulario donde el usuario introduce el nombre del virtual host que desea eliminar. 5.3 El usuario digita un nombre que no existe. 5.4 El usuario presiona el botón eliminar. 5.5 El sistema muestra una notificación de que no existe ningún virtual host con ese nombre.

Con la aplicación de la estrategia de pruebas diseñada se obtuvieron resultados satisfactorios, demostrando el correcto funcionamiento de todas las funcionalidades implementadas.

Se implementaron las pruebas internas, definidas como disciplina de la metodología AUP-UCI, que guía la presente investigación. El método de casos de prueba aplicado demostró resultados satisfactorios desde el

punto de vista funcional. Las no conformidades encontradas fueron analizadas y corregidas en el tiempo establecido, logrando un correcto comportamiento ante diferentes situaciones (entradas válidas y no válidas).

En la primera iteración se detectaron un total de 10 No conformidades: 7 de ortografía, 2 de validación y 1 funcional. En la segunda iteración fueron detectadas 4 No conformidades: 3 de ortografía y 1 de validación. En la tercera no se detectó ningún tipo de error. A continuación, se muestran las no conformidades detectadas en cada iteración de prueba:

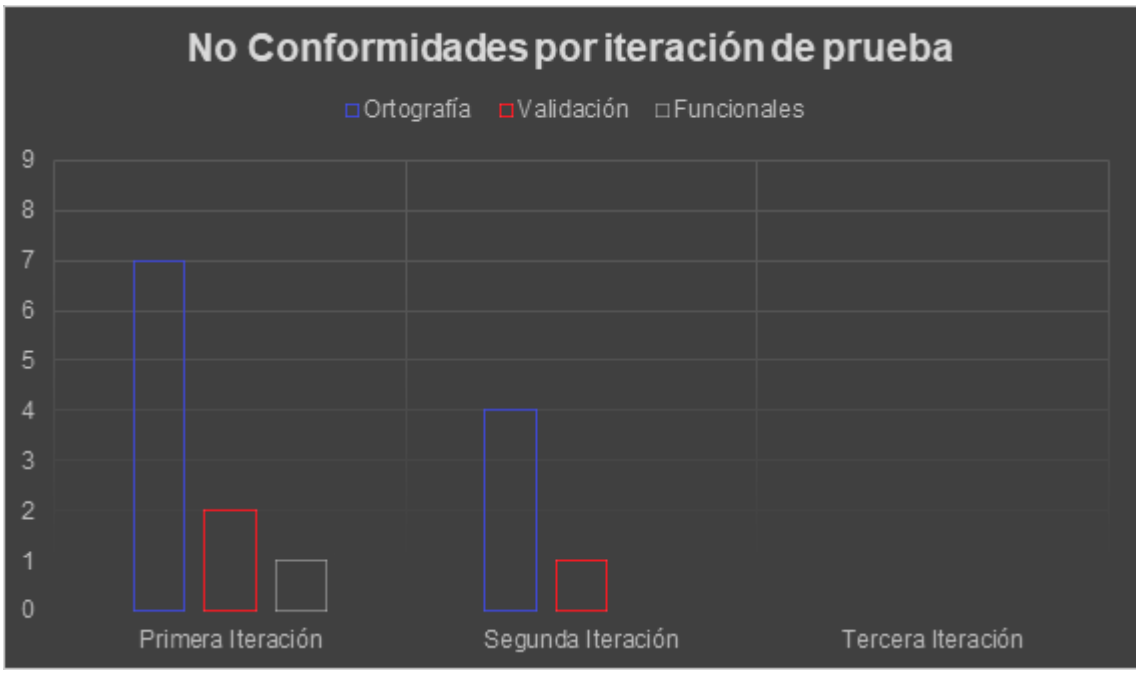


Figura 24 Gráfica que muestra las No conformidades por cada iteración

(Fuente: Elaboración propia)

3.3.3 Pruebas de Aceptación

En el caso de las pruebas de aceptación se aplican las de validación según la estrategia planteada por (Pressman R. S., 2005), con el objetivo de validar que el sistema cumple con el funcionamiento esperado y permitir que el cliente determine su aceptación desde el punto de vista de su funcionalidad y rendimiento. Tienen el objetivo de verificar que el software está listo y que puede ser usado por usuarios finales para ejecutar aquellas funciones y tareas para las cuales fue construido.

Por el cumplimiento de las pruebas de aceptación, el cliente realizó una revisión al sistema con el objetivo de verificar la implementación y el funcionamiento de cada uno de los requisitos funcionales definidos y llegando a la conclusión de que el software cumple con los mismos.

3.4 Conclusiones Parciales

- El adecuado uso de los estándares de codificación permitió desarrollar un código homogéneo, entendible y estandarizado.
- Con la ejecución de las diferentes pruebas realizadas sobre el módulo, arrojó como resultado que este se puede integrar satisfactoriamente en la plataforma Nova ARST, y que este funciona correctamente.

Conclusiones

Al culminar la presente investigación se puede concluir que:

- El estudio de los conceptos asociados a la investigación, el análisis de las diferentes aplicaciones existentes que realizan la configuración del servidor web Nginx, el estudio de la plataforma Nova ARST y la elección de las herramientas tecnológicas y lenguajes de programación que se emplean en el desarrollo del módulo, permitieron definir los umbrales con los que trabaja la aplicación.
- La identificación de los requisitos tanto funcionales y no funcionales, descritos haciendo uso de las historias de usuarios y la elaboración de los artefactos propuestos por la metodología de desarrollo de software empleada permitió facilitar significativamente la codificación de la solución.
- La implementación de la herramienta de administración del servidor web Nginx en la plataforma Nova ARST permitió dar respuesta a los requisitos definidos y de esta manera resolver el problema de investigación planteado.
- El diseño y aplicación de las pruebas a la solución posibilitó la identificación de No conformidades que fueron resueltas satisfactoriamente, logrando así un correcto funcionamiento y la aceptación por parte del cliente.

Recomendaciones

El autor recomienda que futuras versiones del software permitan que se puedan realizar más configuraciones del servidor web Nginx en la plataforma Nova ARST, como la configuración del puerto de escucha para usar este servidor como proxy inverso.

Referencias Bibliográficas

Arevalo, M. (2012). Propuesta de Estándar de desarrollo o codificación (Primera Entrega). Recuperado el 30 de Septiembre de 2022, de https://arevalomaria.wordpress.com/2012/11/02/propuesta-de-estandar-de-desarrollo-o-codificacion-primera-entrega-programacion/.

Arias, M. (2005). La ingeniería de requerimientos y su importancia en el desarrollo de proyectos de software. *Revista de las Sedes Regionales*, 1-13. Recuperado el Agosto de 2022

Banco Santander. (2022). *Santander | Becas*. Recuperado el junio de 2022, de https://www.becas-santander.com/es/blog/python-que-es.html

Burgos, J. (3 de noviembre de 2016). *opencloud Docs*. Recuperado el junio de 2022, de https://docs.opencloud.cl/tutoriales/servidores/introduccion-a-nginx.html

Camilo, S. A. (30 de enero de 2012). *blogs EOI.* Recuperado el 30 de junio de 2012, de http://www.eoi.es/blogs/solangelitacamilo/2012/01/30/el-internet-y-el-mundo/

CVOSOFT. (25 de abril de 2019). *CVOSOFT IT ACADEMY. Obtenido de Plataformas*. Recuperado el 2021, de http://www.cvosoft.com/sistemas_sap_abap/curso_iniciacion_a_sap.php

de la Fuente, C. (s.f.). *Datos*. Recuperado el junio de 2022, de https://datos.gob.es/sites/default/files/doc/file/guia_publicacion_apis.pdf

de Souza, I. (14 de junio de 2019). *rockcontent.* Recuperado el marzo de 2022, de https://docer.com.ar/doc/sc0csec

django. (2022). Recuperado el junio de 2022, de https://www.djangoproject.com

Douglass, P. (2016). *Agile Systems Engineering.*

Fajardo, A. H. (2019). Método basado en la programación por capas para generar código automático desde el diagrama de clases. *Revista PeRuana de ComPutaCión y sistemas*, 25-42. Recuperado el 8 de 2022

Gallardo, J. E., & García, C. M. (2007). *Diseño modular.* Universidad de Málaga, Málaga. Recuperado el Agosto de 2022, de http://www.lcc.uma.es/personal/pepeg/mates

Gonzáles, F., & Vera, Y. (2016). *Módulo de HMAST para la administración y migración hacia SAMBA 4 del servicio directorio activo.* Tesis para optar por el título de Ingeniero en Ciencias Informáticas, Universidad de las Ciencias Informáticas, La Habana. Recuperado el mayo de 2022

Gonzáles, L. (2019). *Propuesta de problemas para prácticas de laboratorio de la asignatura Servicios telematicos.* Trabajo de diploma, Santa Clara. Recuperado el Agosto de 2022

GOV.CO. (2021). *Servicios de Valor Agregado y Telemáticos.* Recuperado el abril de 2022, de http://www.mintic.gov.co/

Guerrero, C., Suárez, J., & Gutiérrez, L. (2013). *Patrones de diseño GOF(The Gang Of Four) en el contexto de Procesos de Desarrollo de Aplicaciones Orientadas a la Web.* Recuperado el 30 de Septiembre de 2022, de https://www.readcube.com/articles/10.4067/s0718-07642013000300012

Gutiérrez, E., & Gato, Y. (Septiembre de 2018). Plataforma para la integración de componentes en un Sistema de Laboratorio Remoto. *Serie Científica de la Universidad de las Ciencias Informáticas*, 33-49. Recuperado el Agosto de 2022

IONOS. (23 de Septiembre de 2020). Recuperado el 30 de Septiembre de 2022, de Digital Guide: https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/diagrama-de-componentes/

Kinsta. (21 de febrero de 2022). *Kinsta.* Recuperado el abril de 2022, de https://kinsta.com/es/base-de-conocimiento/que-es-nginx/

Kotonya, G., & Summerville, I. (1996). *Requirements Engineering with viewpoints.* Recuperado el 3 de julio de 2022, de https://digital-library.theiet.org/content/journals/10.1049/sej.1996.0002

Krall, C. (2006). *Aprender a programar.* Obtenido de http://www.aprenderaprogramar.com/index.php

Kunda, D., Chihana, S., & Sinyinda, M. (2017). Web Server Performance of Apache and Nginx: A Systematic Literature Review. *Computer Engineering and Intelligent Systems*, 43-52.

Labrada, E., & Aragón, Y. (2013). *Desarrollo del Módulo de Gestión de Reportes Estadísticos para el sistema AiresProxyAudit.* La Habana. Recuperado el 30 de Septiembre de 2022

LARDINOIS, F. (2015). *Microsoft shocks the world with visual studio code a free code editor for OSX Linux and windows.* Recuperado el mayo de 2022, de https://www.techcrunch.com/2016/04/14/microsoft-visual-studio-code-for-os-x-linux-and-windows

LARMAN, C. (1999). Introducción al análisis y diseño orientado a objetos. *UML y Patrones*.

Linux, Proyectos. (21 de junio de 2018). *Instalar servidor Nginx, y configurar virtual host en UBUNTU 18.04.* Recuperado el abril de 2022, de https://www.todavianose.com/category/proyectos/

Linux-Console.net. (2019-2022). Recuperado el mayo de 2022

Luján, S. (2002). *Programación de aplicaciones web: historia, principios básicos y clientes web.* Alicante, España: Editorial Club Universitario. Recuperado el junio de 2022, de https://creativecommons.org/licenses/by-nc-nd/3.0/es/legalcode.es

Martínez, N. (2021). *Módulo para la configuración del sistema de archivos en Nova RSAT.* Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas, Universidad de las Ciencias Informáticas, La Habana. Recuperado el mayo de 2022

MDN web docs. (2022). Recuperado el junio de 2022, de https://developer.mozilla.org/es/docs/Web/JavaScript

Mejía, J. T., Gonzáles, M. I., & España, A. (2018). Programming Algorithms of load balancing with HA-Proxy in HTTP services. *Journal of Science and Research: Revista Ciencia e Investigación*, 100-105.

Menzinsky, A., López, G., Palacio, J., Sobrino, M., Álvarez, R., & Rivas, V. (2022). *Historias de Usuario. Ingeniería de Requisitos Ágil.* Recuperado el Agosto de 2022, de https://scrummanager.net/files/scrum_manager_historias_usuario.pdf

Molina, R. (2017). *Módulo para administrar el servidor web Nginx desde la Herramienta para la Migración y Administración de Servicios Telemáticos.* Trabajo de Diploma para optar por el Título de Ingeniero en Ciencias Informáticas, Universidad de las Ciencias Informáticas, La Habana, Cuba. Recuperado el 2022

Montes de Oca, Y., & Brito, Y. (2012). *La gestión de información de trámites protocolizables complejos.* Recuperado el Septiembre de 2022, de https://www.eumed.net/libros-gratis/2012b/1232/arquitectura-N-capas.html

Nedelcu, C. (2013). *Nginx HTTP Server* (Segunda ed.). Birmingham: Packt Publishing. Recuperado el 2022

Nedelcu, C. (2015). *Nginx HTTP Server* (Tercera ed.). Birmingham: Packt Publishing.

Netcraft. (2022). *January 2022 Web Server Survey*. Recuperado el 21 de febrero de 2022, de https://news.netcraft.com/archives/category/web-server-survey/

Ninasunta, M. M., & Pisco, A. S. (2021). *Desarrollo de una plataforma web para la gestión de ofertas laborales a beneficio de los profesionales de la Universidad Técnica de Cotopaxi Extensión La Maná.* Proyecto de investigación presentado previo a la obtención del Título de Ingeniería en Informática y Sistemas Computacionales, Universidad Técnica de Cotopaxi Extensión La Maná, La Maná. Recuperado el junio de 2022, de http://repositorio.utc.edu.ec/bitstream/27000/8213/1/UTC-PIM-000371.pdf

Palma, N. (2013). *Módulo para la administración de los servidores web en HMAST.* Tesis para optar por el título de Ingeniero en Ciencias Informáticas, Universidad de las Ciencias Informáticas, La Habana. Recuperado el 2022

Palma, N. (2019). *Solución informática para la selección de Apache 2 y Nginx durante la migración a código abierto.* Trabajo final presentado en opción al título de Máster en Informática Avanzada, Universidad de las Ciencias Informáticas, La Habana.

Pressman, R. (2002). *Ingeniería de software: Un enfoque práctico.*

Pressman, R. (2002). *Ingeniería de software: Un enfoque práctico 5ta edicion* (Quinta ed.). Nueva York, Estados Unidos: M. Hill.

Pressman, R. S. (2005). *Ingeniería de Software: Un enfoque práctico.* Mexico: McGrawHill.

raiolanetworks. (12 de octubre de 2014). Recuperado el mayo de 2022, de https://raiolanetworks.es/

Ramírez, F. (2018). Metodologías para el desarrollo de software. *Academia*. Recuperado el junio de 2022, de https://www.academia.edu/9953322/Metodologias_para_el_desarrollo_de_software

React. (2022). Recuperado el junio de 2022, de https://es.reactjs.org/

Ricardo, R. (10 de diciembre de 2020). *Plataformas Informáticas: definición, tipo y ejemplos.* Obtenido de file:///D:/UCI/Tesis/Bibliografia/%E2%96%B7%20Plataformas%20inform%C3%A1ticas%20definici%C3%B3n,%20tipos%20y%20ejemplos%20-%20Estudyando.htm

Rodríguez, A., & Pérez, A. (2017). Métodos científicos de indagación y de construcción del conocimiento. *Revista Escuela de Administración de Negocios*. Recuperado el Agosto de 2022

Rodríguez, L., & Roggero, P. (2014). La modelización y simulación computacional como metodología de investigación social. *Polis Revista Latinoamericana*. Recuperado el Agosto de 2022, de http://dx.doi.org/10.4067/S0718-65682014000300019

Saavedra, J. (2007). *Lenguajes de Programación.* Recuperado el mayo de 2022, de https://jorgesaavedra.wordpress.com/2007/05/05/lenguajes-de-programacion/

Sánchez, T. (2015). *Metodología de desarrollo para la Actividad productiva de la UCI.* Universidad de las Ciencias Informáticas, La Habana. Recuperado el mayo de 2022

Solvetic. (4 de mayo de 2014). *Solvetic.* Recuperado el abril de 2022

Somerville, I. (2005). *Ingeniería de Software.* Recuperado el 3 de julio de 2022

Sommerville, I. (2011). *Ingeniería del Software.* (Novena ed.). Mexico: Addison Wesley. Pearson Education. Recuperado el 30 de Septiembre de 2022, de https://www.google.com/search?source=hp&ei=IdIwXo70N82E5wK58rCQDw&q=SOMMERVILLE%2C+I.%2C+2011.+Ingenier%C3%ADa+del+Software.+9na.+Mexico%3A+Addison+Wesley.+Pearson+Education%2C+Inc.+ISBN+978-607-32-0603-7&oq=SOMMERVILLE%2C+I.%2C+2011.+Ingenier%C3%ADa+del+

Soni, R. (2016). *From Begginer to Pro.* New York: Apress.

techopedia. (2022). *¿Qué es un módulo?* Recuperado el 2022, de https://tl.theastrologypage.com/module

Velásquez, L. (28 de mayo de 2021). *Consultora Inicial.* Obtenido de file:///D:/UCI/Tesis/Bibliografia/%C2%BFQu%C3%A9%20es%20un%20software%20de%20administraci%C3%B3n%20remota%20-%20Consultora%20Inusual.htm

Vilajosana, X., & Navarro, L. (2019). *Arquitectura de aplicaciones web.* Universitat Oberta de Catalunya, Catalunya. Recuperado el Septiembre de 2022, de http://190.57.147.202:90/jspui/bitstream/123456789/465/1/Arquitectura-de-aplicaciones-web-M2.pdf

W3Techs. (2022). *Usage statistics of web servers.* Recuperado el 21 de febrero de 2022, de https://w3techs.com/technologies/overview/web_server

Anexos

Anexo 1. Entrevista realizada al Ing. Lexys Manuel Díaz para determinar las características de la plataforma Nova ARST.

Teniendo en cuenta sus conocimientos sobre la plataforma responda:

1. ¿Qué es y cómo funciona la plataforma Nova ARST?
2. ¿Con qué arquitectura está diseñada la plataforma?
3. ¿Qué lenguajes o que tecnologías fueron empleadas en su desarrollo?
4. ¿Qué elementos se deben tener en cuenta para poder integrar un módulo en la plataforma Nova ARST?

Anexo 2. Entrevista realizada a varios trabajadores del nodo central de la UCI para una mejor obtención de requisitos.

1. ¿En la universidad se utiliza el servidor web Nginx?
2. ¿Se utiliza alguna herramienta para realizar la configuración de dicho servidor?
3. ¿Cuáles son las principales configuraciones que se realizan sobre el servidor?

Anexo 3. Historias de usuarios

Tabla 12 Historia de Usuario 5

(Fuente: Elaboración propia)

Número: HU – 5	**Nombre del requisito:** Reiniciar el servidor web Nginx
Programador: Leovaldo Pérez Rojas	**Iteración Asignada:** 1
Prioridad: Alta	**Tiempo Estimado:**1 semanas
Riesgo en Desarrollo: Medio	**Tiempo Real:** no definido
Descripción: El sistema debe permitir al usuario poder reiniciar el servidor Nginx una vez que este se haya iniciado previamente en el sistema. • El usuario accede al sistema.	

- El usuario selecciona la opción reiniciar haciendo *click* en el botón correspondiente

Observaciones: se ejecuta en la pc servidora el comando service nginx restart

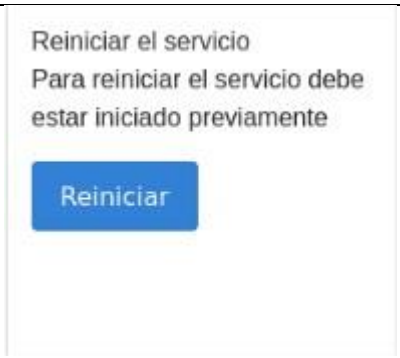


Tabla 13 Historia de Usuario 7

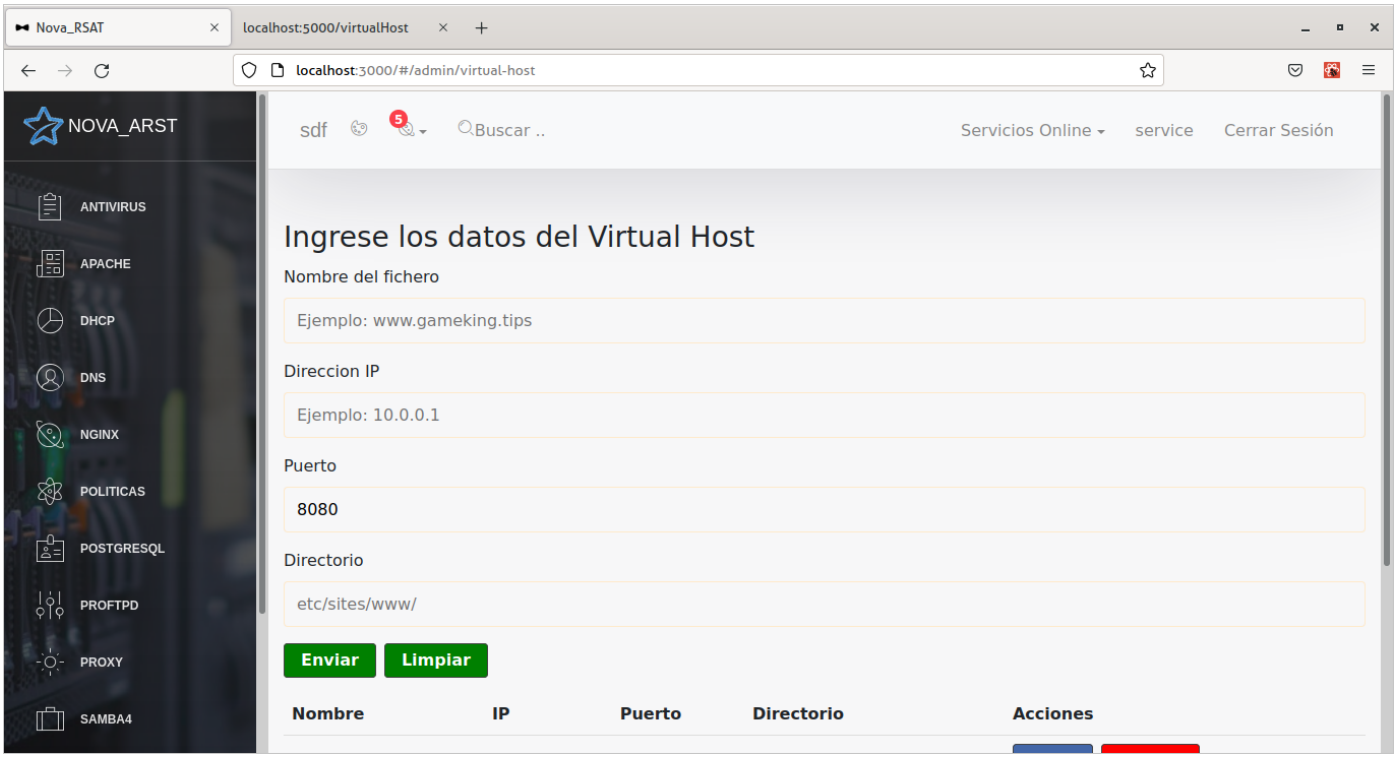
(Fuente: Elaboración propia)

Número: HU – 7	**Nombre del requisito:** Modificar virtual host
Programador: Leovaldo Pérez Rojas	**Iteración Asignada:** 1
Prioridad: Media	**Tiempo Estimado:**1 semanas
Riesgo en Desarrollo: Medio	**Tiempo Real:** no definido

Descripción: El sistema debe permitir al usuario poder modificar los parámetros de un virtual host que haya sido añadido al sistema previamente.

- El usuario accede al sistema.
- El usuario presiona el botón opciones de la interfaz principal.
- El sistema muestra un formulario con el campo nombre del fichero.
- El usuario escribe el nombre del fichero y presiona el botón Editar.
- El sistema debe mostrar un formulario con el campo nombre de fichero. Los campos a rellenar son:

- Nombre del fichero: indica el nombre que se le dará al fichero que contiene la configuración del virtual *host*. Es un campo obligatorio y no debe tener más de 255 caracteres.
- Dirección IP: indica la dirección IP del virtual *host* en caso de existir más de una interfaz de red. Se corresponde con el primer argumento de la directiva ***listen***. Es un campo obligatorio. Puede tener varios valores: si la dirección IP es un (*) se asume que se puede acceder a un virtual *host* desde cualquier dirección IP con un puerto determinado. Cuando la directiva ***listen*** aparece solo con el valor del puerto, por defecto se asume la dirección IP del servidor.
- Puerto: se refiere al puerto por el que escucha el servidor asignado al virtual *host*, que debe tener un valor entero positivo entre 1 y 65535. Este responde a la segunda parte del argumento de la directiva ***listen***. Es un campo obligatorio. El puerto puede ser un (*) en caso de que se defina acceder a un virtual *host* mediante una dirección especificada por cualquier puerto.
- Directorio raíz: indica el directorio que contiene lo que será publicado por el virtual *host*. Debe ser la ruta de un directorio que exista en el servidor. Se corresponde con el argumento de la directiva ***root***. Es un campo obligatorio.
- Nombres del servidor: indica la lista de nombres de dominio que se asignan al virtual *host*. Cada elemento debe ser un nombre de dominio válido. Se corresponde con el argumento de la directiva ***server_name***. No es un campo obligatorio.


Nova_RSAT
localhost:5000/virtualHost
localhost:3000/#/admin/virtual-host
NOVA_ARST
ANTIVIRUS
APACHE
DHCP
DNS
NGINX
POLITICAS
POSTGRESQL
PROFTPD
PROXY
SAMBA4
sdf
Buscar ..
Servicios Online
service
Cerrar Sesión
Ingrese los datos del Virtual Host
Nombre del fichero
Ejemplo: www.gameking.tips
Direccion IP
Ejemplo: 10.0.0.1
Puerto
8080
Directorio
etc/sites/www/
Enviar
Limpiar
Nombre
IP
Puerto
Directorio
Acciones

Glosario de términos

FastCGI: es un protocolo para interconectar programas interactivos con un servidor web. Es una variación de Interfaz Común de Entrada (CGI, del inglés *Common Gateway Interface*).

GNU/Linux: sistema operativo, más conocido como Linux.

Virtual host: se refiere a que en un mismo servidor web se pueden hospedar múltiples proyectos cada uno con su propio dominio, aunque todos pertenezcan a la misma dirección IP pública.

Netcraft: es una compañía de servicios de internet que ofrece análisis de cuota de mercado de servidores y alojamiento web.

Servidor web: o servidor HTTP es un programa informático que procesa cualquier aplicación del lado del servidor.

Sites-available: directorio que contiene los sitios disponibles en Nginx.

Sites-enabled: directorio que contiene los sitios habilitados en Nginx.