Universidad de las Ciencias Informáticas

Facultad 1

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

Componente de Visualización de trazas para el sistema Xilema Smart Keeper 3.0

Autor: Luis Gabriel Fernández-Vega Rojas

Tutores: Ing. Rubén Roberto Peña Domínguez
Ing. Yordanka Fuentes Castillo

La Habana, junio de 2018

Declaración de autoría

Yo, Luis Gabriel Fernández-Vega Rojas, declaro ser el autor del presente trabajo de diploma y reconozco a la Facultad 1 de la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo. Autorizo a dicha institución universitaria para que haga el uso que estime pertinente de este trabajo.

Para que así conste firman los presentes a los ____ días del mes de _____ del año _____.

Luis Gabriel Fernández-Vega Rojas

_____

Autor

Ing. Rubén Roberto Peña Domínguez

_____

Tutor

Ing. Yordanka Fuentes Castillo

_____

Tutor

Resumen

El siguiente trabajo tiene como objetivo proponer un Componente de Visualización de trazas para el sistema Xilema Smart Keeper 3.0, que permita a los usuarios consultar información detallada referente a su navegación. El componente permite que en la nueva versión del sistema Xilema Smart Keeper se puedan visualizar las trazas de navegación y consultar la cuota de sus usuarios. Puesto que la versión actual del sistema Xilema Smart Keeper no cuenta con una forma de visualizar esta información, siendo la misma de gran utilidad para los usuarios que poseen servicio de *internet,* se plantea como objetivo el desarrollo de dicho componente*.* Para darle cumplimiento al objetivo planteado en la investigación se realizó un estudio del estado del arte sobre los componentes de visualización de trazas. Durante la investigación se analizaron los sistemas homólogos a nivel nacional e internacional definiendo los principales aspectos que pudieran emplearse en el desarrollo del componente. El proceso de desarrollo estuvo guiado por la metodología de desarrollo de *software* AUP-UCI, y se emplearon como principales tecnologías: *Symfony* 3.4.9 como marco de trabajo, PHP 7, HTML 5 y *JavaScript* como lenguajes de programación, el motor de búsqueda Elasticsearch para el almacenamiento de los *logs* y *Visual Paradigm* 10.1 como herramienta para el modelado. A partir de la aplicación de las pruebas de *software* de tipo funcional, de integración y de carga y estrés se constató que el componente implementado permite la correcta visualización del consumo de cuota del usuario autentificado, de sus trazas de navegación, así como filtrar estas últimas por diferentes parámetros.

Palabras clave: consulta de cuota, información detallada, *logs*, visualización de trazas de navegación

Índice

Índice de figuras

Índice de tablas

Introducción

En la actualidad es innegable que desde el surgimiento de las Tecnologías de la Información y la Comunicación (TIC), la informática se ha convertido en una de las ciencias vanguardistas a nivel mundial, siendo el *internet* uno de los mayores exponentes en lo que a flujo de información se refiere. Esto ha permitido el intercambio de ideas y conceptos, la sincronización de proyectos y eventos, e infinidades de nuevas posibilidades que ponen al alcance, tanto de individuos como de corporaciones y empresas, una herramienta capaz de mejorar su estilo de vida, condiciones de trabajo y su productividad. Sin embargo, con tanta información se hace necesario crear una forma de optimizar su uso para reducir los costos en tiempo y dinero generados por el consumo de estos recursos.

Según Magdalena Claro, si bien estas tecnologías tienen en común la manipulación y comunicación de información en formato digital, sus aplicaciones, funciones y características son muy diversas. Por otra parte, las TIC son instrumentos, y como tales, pueden ser usados de muy distintas formas (Claro, 2010).

A mayor desarrollo social y tecnológico, mayor capacidad de generar datos e informaciones y también mayor dificultad para seleccionarlos, categorizarlos y memorizarlos. Es en este punto cuando los componentes de visualización demuestran su importancia permitiendo a usuarios y administradores desplazarse a través de una interfaz, mediante la cual surge la posibilidad de controlar de manera más fácil e intuitiva la tecnología a la que estén accediendo o administrando. Todo en pos de la comodidad del usuario y con el fin de alcanzar una mayor eficiencia y permitir un mejor entendimiento de la actividad realizada por la vía de los medios de la informática y la comunicación (IV Congreso, 2009).

Juan Carlos Dürsteler López define a la visualización de la información como “proceso de interiorización del conocimiento mediante la percepción de la información o, si se quiere, mediante la elaboración de datos” (Dürsteler, 2003, p.22).

Realmente se hace necesario construir una estructura que facilite el entendimiento de lo que se quiere transmitir y de lo que se recibe. Y esta organización que está profundamente unida al diseño de los sitios *web*, converge como pieza clave para la transformación de la información en conocimiento (Jiménez y Meseguer, 2007, p.139).

Cabe mencionar que en *internet* se encuentran grandes volúmenes de contenidos de diversas fuentes y temáticas que pueden variar desde materiales educativos, de ocio y científico-técnico hasta los considerados inadecuados (nocivos e ilícitos) (García, 2005). Por otra parte, comúnmente instituciones que ofrecen servicio de *internet* establecen una Política de Uso Aceptable de *Internet* (en lo adelante AUP, por sus siglas en inglés) propia, constituida por un conjunto de reglas que restringen las formas en que los sistemas informáticos pueden ser usados (Fernández, 1994). En ocasiones, el establecimiento de estas regulaciones no significa medida suficiente para su cumplimiento. En este sentido, los filtros de contenido emergen como una solución informática que contribuyen al cumplimiento de las normas establecidas de navegación.

Como parte de uno de los diversos programas implementados en Cuba por la batalla de ideas, y específicamente durante el año 2002, es creada la Universidad de las Ciencias Informáticas (UCI). La misma es reconocida no solo por ser un centro docente sino también por ser un centro productor de *software*. Para ello cuenta con 14 centros de desarrollo entre los que se encuentra el Centro de Ideoinformática (CIDI), que a su vez cuenta entre otros con el departamento Soluciones Informáticas para *Internet* (SINI) (UCI, 2017).

Uno de los sistemas desarrollados por este departamento es Xilema Smart Keeper; el mismo es una *suite* de gestión del acceso a *internet* y consiste en un sistema de filtrado y dos analizadores de *logs* de servidores *proxys* que permite aplicar las AUP o, en otras palabras, el control de acceso a contenidos de *internet* en una institución.

Su interfaz de administración es la que simplifica este mismo proceso por parte de los administradores además de la generación de datos estadísticos. Por otra parte el Quota_contrain es un componente del subsistema de filtrado que permite contar la cuota de navegación en datos y tiempo de conexión (Manual de Usuario Smart Keeper, 2014).

El sistema Xilema Smart Keeper actual fue desarrollado para desplegar en servidores con sistema operativo Ubuntu 12.04. Debido a que hasta la fecha han salido nuevas versiones del sistema operativo, quedando la 12.04 obsoleta, el sistema Smart Keeper vigente también queda obsoleto. Por esta razón se desea desarrollar una nueva versión usando para ello tecnologías más actuales.

Dentro de los componentes con que cuenta la versión actual de Xilema Smart Keeper, se encuentra el componente para visualizar el consumo de la cuota de navegación de los usuarios de la entidad donde se encuentre desplegado el sistema. Al migrar a la nueva versión de Xilema Smart Keeper este módulo quedará en desuso, por lo cual los usuarios pertenecientes a las empresas donde se despliegue la nueva versión, no podrán visualizar su consumo de cuota a través del sistema. Ello traería como consecuencia que el sistema no les brinde a los usuarios de la entidad este servicio, por lo que no les sería posible conocer con exactitud cuánto le queda por consumir de su cuota asignada.

Además de los problemas ya mencionados que se derivan de la incompatibilidad entre las tecnologías de la nueva versión de Smart Keeper y el componente de visualización vigente, este último presenta otras carencias en cuanto a su funcionalidad. El mismo no permite mostrar las trazas por la dirección IP a la cual se accedió, ni por la fecha en que se hizo la navegación, ni el dominio al que pertenece la página consultada. Estos datos resultan ser de vital importancia para la planificación personal en cuanto a la administración individual de su cuota o en caso de una violación de las normas del uso debido de la cuota de navegación asignada.

Teniendo en cuenta la situación problémica descrita anteriormente se plantea el siguiente **problema de investigación**: ¿Cómo facilitarles a los usuarios del sistema Xilema Smart Keeper 3.0 la información referente a su navegación y cuota consumida?

Con el objetivo de encaminar la investigación se identifica como **objeto de estudio** el proceso de visualización de trazas en sistemas *web*, el cual enmarca al **campo de acción**: componente de visualización de trazas para el sistema Xilema Smart Keeper.

Para dar solución al problema descrito, se ha planteado el siguiente **objetivo general**: Desarrollar un componente de visualización de trazas para el sistema Smart Keeper 3.0 que permita a los usuarios consultar información detallada referente a su navegación y cuota consumida.

Para dar cumplimiento al objetivo general se plantean los siguientes **objetivos específicos:**

- Describir el marco teórico conceptual y el estado del arte referente a las herramientas de gestión de acceso a *internet* y los componentes de visualización.

- Analizar y diseñar el componente para la visualización de trazas del sistema Xilema Smart Keeper 3.0.
- Implementar el componente para la visualización de trazas del sistema Xilema Smart Keeper 3.0.
- Validar el *software* desarrollado.

Ya planteados los objetivos a vencer en el presente trabajo y las áreas de la ciencia en las cuales se quieren materializar los mismos, se plantea la siguiente **idea a defender:** El desarrollo de un componente de visualización de trazas para el sistema Xilema Smart Keeper 3.0 permitirá a los usuarios visualizar la información correspondiente a su navegación y cuota consumida.

Para lograr los objetivos anteriormente descritos se plantean las siguientes **tareas de investigación:**

1- Revisión de la bibliografía existente sobre los conceptos asociados al marco teórico de la investigación.
2- Aclaración de los conceptos asociados al dominio del problema.
3- Estudio de los sistemas homólogos existentes en Cuba y el mundo.
4- Selección de las herramientas, tecnologías y metodología.
5- Modelación del negocio.
6- Identificación y documentación de los requisitos funcionales y no funcionales del sistema.
7- Diseño del componente a desarrollar acorde a la metodología de desarrollo, patrones y arquitectura a emplear.
8- Desarrollo de las funcionalidades del componente.
9- Diseño y ejecución de las pruebas de *software* que serán aplicadas para la adecuada validación de la solución presentada.

Posibles Resultados: Se espera obtener un componente de visualización de trazas que permita a los usuarios visualizar, por diferentes parámetros, el consumo de las cuotas asignadas a través del sistema Xilema Smart Keeper 3.0.

Para guiar el proceso de investigación del presente trabajo se utilizaron los siguientes métodos científicos:

Métodos teóricos

- **Histórico-lógico:** Se utilizó para el estudio del desarrollo de la información escalando por las TIC hasta llegar a los componentes de visualización y *software* de control de acceso a *internet*.
- **Analítico-Sintético:** Se utilizó al analizar la teoría existente sobre los componentes de visualización en especial para interfaces visuales.
- **Modelado:** Se utilizó para realizar una reproducción simplificada de la realidad, mostrando las relaciones internas y características del sistema a través de diagramas.

Métodos empíricos

- **Entrevista:** Se empleó para recopilar información referente a los requisitos funcionales y no funcionales con los que deberá cumplir la propuesta de solución.

El siguiente trabajo de diploma se estructura de la siguiente forma: Introducción**,** tres capítulos, conclusiones y bibliografía.

El Capítulo 1 titulado: “Fundamentación teórica de los Componentes de Visualización de trazas” ofrece los conceptos básicos asociados al dominio del problema. Además, brinda el estado del arte referido al tema de investigación, así como las técnicas, tecnologías, metodologías y herramientas usadas para el desarrollo de la solución.

El Capítulo 2 titulado “Análisis y diseño de la propuesta de solución Componente de Visualización de trazas para el Sistema Xilema Smart Keeper 3.0” menciona los requisitos funcionales y no funcionales del componente, además de las historias de usuario del sistema. Propone una arquitectura de *software*, así como los patrones de diseño a emplear. Se muestran los distintos diagramas que describen el funcionamiento del componente.

En el Capítulo 3 titulado “Implementación y validación del Componente de Visualización de trazas para el sistema Xilema Smart Keeper 3.0” se muestra el diagrama de componentes y su descripción, con el fin de comprender los elementos de *software* del subsistema y sus relaciones. También se expone la validación del diseño y el componente implementado mediante la ejecución de un conjunto de pruebas.

Capítulo 1: Fundamentación teórica de los Componentes de Visualización de trazas

1.1 Introducción

En el presente capítulo se presenta un panorama general de los principales aspectos relacionados con los componentes de visualización. Se listan las características de varias herramientas y las definiciones de principal importancia asociadas al dominio del problema con el propósito de entender la solución propuesta. Se definen los principales conceptos a emplear en la investigación y se hace una revisión de aplicaciones homólogas tanto en el ámbito nacional como en el internacional, así como una comparación entre las mismas. Además, se realiza la selección del ambiente de desarrollo con el cual se implementará la solución, partiendo desde la metodología para guiar este proceso, hasta las herramientas y lenguajes a emplear.

1.2 Visualización de datos

La visualización de datos es un término general que describe cualquier esfuerzo para ayudar a las personas a comprender la importancia de los datos, colocándolos en un contexto visual. Los patrones, las tendencias y las correlaciones que pueden pasar desapercibidos en los datos basados en texto, pueden exponerse y reconocerse más fácilmente con un *software* de visualización de datos (TechTarget*,* 2017).

“En la actualidad nuestro consumo de información se ha multiplicado de manera exponencial debido a dos factores: cada vez se produce más información (redes sociales, dispositivos, etc.) y cada vez tenemos más capacidad de acceso a dicha información, especialmente a través de *Internet* y de la *Web*. La capacidad de sacar partido y entender la información bruta está íntimamente ligada a nuestra capacidad para explotarla y transformarla en algo más que puro dato: los datos adquieren significado” (Datos, 2017).

No obstante, los datos en sí, como registros aislados, no aportan un significado concreto. Sólo cuando estos son interpretados, es que cobran sentido y se transforman en conocimiento. En el ámbito tecnológico, la explotación de datos ha evolucionado en las últimas décadas implementando mecanismos de interpretación cada vez más robustos y asequibles. Y, entre estos mecanismos, el más importante es la visualización de datos (Datos, 2017).

1.3 Trazas de navegación

Una traza es una captura de todo el tráfico de red que se envía y se recibe entre dos o más puntos de la red (un teléfono, un ordenador, un *router*, etc.) de forma que se pueda ver qué se envía y se recibe. Hay muchos tipos de trazas: por las trazas de red se ven todos los protocolos, todos los datos, con una traza SIP (en español, Protocolo de Iniciación de Sesión) se filtra todo el tráfico y se obtienen únicamente los paquetes SIP, o de cualquier otro protocolo. Dado que una traza permite ver qué se ha enviado y qué se ha recibido, esto suele ayudar a entender el comportamiento de los usuarios en la red (SinoLogic, 2017).

1.4 Servidor

Un servidor es un programa de aplicación que acepta conexiones con el objetivo de dar servicios a solicitudes mediante el envío de regreso de respuestas. Los términos cliente y servidor hacen referencia a los roles cumplidos por un programa en una conexión dada, pudiendo cualquier programa actuar tanto como cliente o servidor frente a distintas conexiones (Ballari, 2002).

1.5 Proxy

Un programa intermediario que actúa como cliente y servidor con el objetivo de realizar solicitudes en nombre de otros clientes. Las solicitudes son resueltas internamente o mediante el reenvío, con una posible traducción, a otros servidores. Según RFC 2616 (*Request for Comments*), un "*proxy* transparente" es aquel que no modifica la solicitud o la respuesta más allá de lo necesario para las acciones de autenticación y/o identificación del *proxy*. Un "*proxy* no transparente" es aquel que modifica la solicitud o la respuesta con el objetivo de proveer algún servicio agregado al cliente. El término "*proxy* transparente" es muchas veces usado incorrectamente para designar "*proxy* interceptor".

1.6 Logs (Registros)

Los *logs* son una lista detallada de la información de una aplicación, el rendimiento del sistema o las actividades del usuario. Un *log* puede ser útil para realizar un seguimiento del uso de una computadora, una recuperación de emergencia o el mejoramiento de aplicaciones. Cada programa de *software* que es capaz de crear un registro tiene diferentes métodos para iniciar o detener la creación de *logs*. Muchas páginas

web mantienen registros de las páginas a las que acceden los usuarios, los errores producidos, cómo es la visibilidad de la página por parte de las personas y mucha más información. Estos *logs* ayudan a los propietarios de la página *web* a realizar un seguimiento de las estadísticas de uso, y les informa si es necesario alguna corrección o actualización en el sitio (Computer Hope*,* 2017).

1.7 Estudios de los sistemas homólogos

En el ámbito internacional

1. Webalizer: Es una herramienta gratuita de análisis de archivos de registro que genera resúmenes de utilización muy detallados acerca de servidores *web* y FTP en formato gráfico y tabular de fácil comprensión. Los registros son un resumen estadístico del tráfico de usuarios en su servidor. Los informes de *Webalizer* se generan en horarios programados de acuerdo con las opciones de configuración que se establecieron para el sitio al momento de crearlo (The Webalizer, 2017).

2. AWStats (*Advanced Web Statistics* o Estadística Web Avanzada): Es una herramienta de análisis open source escrita en *Perl*, que permite parsear *logs* de un servidor *web*, ftp o de correo y generar estadísticas en forma de reportes HTML. Puede ejecutarse desde un navegador *web* o desde la línea de comandos y funciona sobre diversos sistemas operativos.

Un análisis del registro completo permite mostrarles la siguiente información: el número de visitas y de visitantes únicos y la duración de las visitas, los usuarios autenticados y últimas visitas autenticadas, las páginas más vistas, los navegadores utilizados (páginas, hits, kb para cada navegador), OS utilizado (páginas, hits, KB para cada sistema operativo), los motores de búsqueda, frases y palabras clave utilizadas para encontrar su sitio y los errores HTTP (Kevell, 2017).

3. Squid Analysis Report Generator (Sarg): Es una herramienta que permite conocer, toda la actividad de los usuarios y/o equipos incluidos en la red, registrada en los *logs* del *proxy*. Fue desarrollada con el lenguaje de programación C y es distribuida bajo la licencia GPL v2. Provee mucha información sobre las actividades de los usuarios de Squid: tiempo, bytes, sitios y otros. Este analizador brinda una variedad de información como: listados de sitios más visitados, reportes diarios, semanales y mensuales, gráficos semanales y

mensuales del consumo por usuario/host, detalles de todos los sitios a los que entró un usuario/host y descargas.

En el ámbito nacional

1. AiresProxy: es un indagador de ficheros de *logs* que brinda a los usuarios la información referente a sus registros de navegación en forma de reportes. Posee un motor de segmentación de registros, otro de mantenimiento a la base de datos y por último un módulo de análisis y presentación de datos. También posibilita actualizar la cuota de navegación de los usuarios en caso de que la institución establezca cuotas para acceder a redes externas.

El sistema posee, además, un programa de mantenimiento auxiliar encargado de eliminar de la base de datos los registros que no son de interés del usuario o los administradores de red. Es una herramienta muy versátil gracias al elevado poder de configuración mediante ficheros XML. Dentro de la información que brinda se puede encontrar: reporte por dominios, reporte de dominio por dirección IP9, reporte de dominio por fecha, reporte por dirección IP, reporte por fecha y reporte por URL (Chavez, 2011).

2. ISAWeb: es un producto realizado en el Instituto Central de Investigación Digital (ICID), el cual le permite a un usuario autenticado poder filtrar información relevante sobre el acceso de *internet*. Además, genera una variedad de reportes y posee filtros para la búsqueda de información. La aplicación valida las credenciales entradas por el usuario contra el servidor *Active Directory* del dominio, mediante el protocolo LDAP. La información de tráfico de *internet* se extrae del servidor de *Internet* Isa *Server*, transformando los ficheros donde se almacenan las trazas de la navegación, a una base de datos en *SQL Server* (Palenzuela, 2006).

3. AAInternet: Aplicación realizada por Segurmática, la cual permite extraer información de los ficheros *log* de los servidores *proxy*, generar reportes gráficos de análisis de la navegación de los usuarios, donde pueden observarse la fecha y hora de visita al sitio, los dominios o sitios *web* visitados y la transferencia por usuarios, entre otros datos. Funciona sobre el sistema operativo *Windows.* Utiliza una base de datos creada a partir del procesamiento de archivos *log* del servidor *proxy*, para ello es necesario copiar manualmente hacia la estación donde está instalada esta herramienta los archivos *log* que se deseen parsear o a una

estación dentro de la red donde sean accesibles estos archivos desde la estación donde se encuentra instalado AAInternet (Aguilera y González, 2017).

4. SRNI+: El Sistema de Reportes de la Navegación por Internet (SRNI) es una aplicación *web* realizada en la UCI, desarrollada bajo la tecnología Java y que utiliza como gestor de base de datos *PostgreSQL*. Debido a que es una aplicación *web* puede consultarse desde cualquier lugar de la UCI. El *software* brinda a los usuarios reportes dinámicos de su navegación, los cuales son creados a partir de las trazas del servidor *proxy*. La información de acceso que se puede consultar es por URL, dirección IP, días y horas (Martín Álvarez y García Martínez, 2007).

1.7.1 Resultado del estudio de sistemas homólogos

Luego de analizar cada una de las herramientas y soluciones existentes, ninguna satisface las necesidades que plantea la institución por las siguientes condiciones: la herramienta que funciona sobre el sistema operativo *Windows* (AAInternet) no se tendrá en cuenta para darle solución al problema por funcionar sobre un SO (*Operative System*) propietario, lo que conlleva a que para su uso haya que pagar una costosa licencia y además, que no se pueda modificar su código fuente para adaptarlo a las necesidades del problema. Aquellas que son *software* libre (AWStats*, Squid Analysis Report Generator y Webalizer*) no están orientadas a usuarios. Las herramientas nacionales AiresProxy, ISAWeb y SRNI+ se enfocan en los usuarios, sin embargo, no es posible utilizar ninguna de estas aplicaciones existentes debido a la incompatibilidad en términos de lenguajes de programación, tecnologías empleadas o uso de herramientas con versiones que han quedado desactualizadas.

Aunque ninguna de las aplicaciones antes mencionadas representa una solución factible para el problema planteado, es necesario destacar que su estudio aportó ideas que deben ser tomadas en cuenta en la solución a implementar, que pueden ser transformadas en requisitos funcionales y no funcionales. Por lo tanto, se evidencia la necesidad de desarrollar un componente que permita la visualización de trazas por parte de los usuarios del sistema Xilema Smart Keeper 3.0.

1.8 Selección de la metodología de desarrollo de software a utilizar

Para seleccionar una metodología de desarrollo adecuada para cualquier proyecto se deben tener en cuenta las características particulares de los mismos. A continuación, se expresan una serie de aspectos por los que se ha seleccionado la **AUP-UCI** como metodología para este proyecto, así como una descripción de la misma:

AUP-UCI es una variación de la metodología ágil AUP, adaptándola al ciclo de vida definido para la actividad productiva en la UCI.

- Describe de una manera simple y fácil de entender la forma de desarrollar aplicaciones de *software* en la UCI, utilizando a su vez las buenas prácticas del modelo CMMI.
- Es una metodología ágil, por tanto, se adapta al trabajo de equipos de pequeño formato.
- Permite la gestión de cambios ágiles sin la necesidad de generar excesiva documentación.
- Es la metodología utilizada en el proyecto Smart Keeper, al cual se integrará el componente final.
- Permite la descripción simplificada del proceso de desarrollo de *software* acortando los tiempos de desarrollo de una aplicación.
- De esta metodología se decide usar el cuarto escenario para la disciplina de requisitos pues el mismo es recomendado para proyectos no muy extensos y que no modelen negocios.

Esta metodología cuenta con 3 fases las cuales se describen como:

Inicio: Es en este punto del proyecto donde se llevan a cabo las actividades relacionadas con la planeación del mismo. En esta fase se definen el alcance del proyecto, las estimaciones de tiempo, el esfuerzo y costo y se decide si el proyecto será ejecutado o no; todo esto a través de un estudio al cliente.

Ejecución: En esta fase se ejecutan las actividades de desarrollo del *software* y se ajustan 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 los resultados del proyecto, se comprueba su correcta ejecución y se realizan las actividades formales de cierre del proyecto.

Las disciplinas se llevan a cabo de manera sistemática con objeto de desarrollar, validar, y entregar un *software* que responda a las necesidades de sus clientes. Estas disciplinas son: (Sánchez, s.f.)

1. Modelado de negocio: En esta disciplina se comprende cómo funciona el negocio que se desea informatizar como garantía de que el *software* desarrollado cumpla con el propósito planteado.
2. Requisitos: La tarea de esta disciplina es desarrollar un modelo del sistema que se va a construir. En la misma se comprenden la administración y gestión de los requisitos funcionales y no funcionales del producto.
3. Análisis y diseño: Para esta parte, de ser necesario, los requisitos pueden ser refinados y estructurados para conseguir una comprensión más precisa de estos, y una descripción que sea fácil de mantener y ayude a la estructuración del sistema. Además, en esta disciplina se modela el sistema y su forma para que soporte todos los requisitos.
4. Implementación: En la implementación se construye el sistema a partir de los resultados del análisis y el diseño.
5. Prueba interna: En esta disciplina se verifica el resultado de la implementación probando cada construcción, incluyendo tanto las construcciones internas como intermedias, así como las versiones finales a ser liberadas. Se deben desarrollar artefactos de prueba como: diseños de casos de prueba, listas de chequeo y de ser posibles componentes de prueba ejecutables para automatizar las pruebas.
6. Pruebas de liberación: Pruebas diseñadas y ejecutadas por una entidad certificadora de la calidad externa, a todos los entregables de los proyectos antes de ser entregados al cliente para su aceptación.
7. Pruebas de aceptación: Es la prueba final antes del despliegue del sistema. Su objetivo es verificar que el *software* está listo y que puede ser usado por usuarios finales para ejecutar aquellas funciones y tareas para las cuales el *software* fue construido.
8. Por último, para la parte de gestión y soporte se definen por las áreas de procesos de CMMIDEV v1.3 para el nivel 2, estas serían CM (Gestión de la configuración), PP (Planeación de proyecto) y PMC (Monitoreo y control de proyecto).

A partir de que el Modelado de negocio propone tres variantes a utilizar en los proyectos (Casos de uso del negocio (CUN), Descripción de proceso de negocio (DPN) o Modelo conceptual (MC)) y existen tres formas

de encapsular los requisitos (Casos de uso del sistema (CUS), Historias de usuario (HU), Descripción de requisitos por proceso (DRP)), surgen cuatro escenarios para modelar el sistema en los proyectos los cuales son:

- Proyectos que modelen el negocio con CUN solo pueden modelar el sistema con CUS.

CUN + MC =CUS

- Proyectos que modelen el negocio con MC solo pueden modelar el sistema con CUS.

MC = CUS

- Proyectos que modelen el negocio con DPN solo pueden modelar el sistema con DRP.

DPN + MC = DRP

- Proyectos que no modelen negocio solo pueden modelar el sistema con HU.

HU

1.9 Ambiente de desarrollo

1.9.1 Lenguajes utilizados

HTML 5

HTML es un lenguaje de computadora ideado para permitir la creación de sitios *web*. Estos sitios *web* pueden ser vistos por cualquiera quien se encuentre conectado a *internet*. Es relativamente fácil de aprender siendo su contenido básico de fácil acceso para la mayoría de las personas; y bastante poderoso en lo que permite crear. Se encuentra en constante revisión y evolución para cumplir con las demandas y requisitos de la creciente audiencia de *internet* bajo la dirección del W3C (*World Wide Web Consortium*), la organización encargada del diseñando y manteniendo el lenguaje. La definición de HTML es *HyperText Markup Language* (Lenguaje de marcado de hipertexto) (Shannon, 2007).

HTML5 provee básicamente 3 características: estructura, estilo y funcionalidad. El mismo es considerado el producto de la combinación de HTML, CSS y *JavaScript*. Estas tecnologías son altamente dependientes y actúan como una sola unidad organizada bajo la especificación de HTML5 (Gauchat, 2012).

CSS 3

El *Cascading Style Sheets* (CSS) es un lenguaje de diseño estándar para la *web*. La clave de su invención por parte de Bert Bos y Hakon Wium Lie, es que CSS separa la presentación de la estructura subyacente y la semántica. CSS saca las instrucciones visuales de HTML y se adhiere lo suficiente para presentar el sitio en la forma de un navegador gráfico tradicional (Lie y Bos, 2005).

JavaScript 1.2

JavaScript es un lenguaje de programación interpretado con capacidades orientadas a objetos (OO). Sintácticamente se asemeja a C, C ++ y Java, con construcciones de programación como la instrucción *if*, el ciclo *while* y el operador &&. La similitud termina con este parecido sintáctico, sin embargo, es un lenguaje de tipo suelto, lo que significa que las variables no necesitan tener un tipo específico. Los objetos en *JavaScript* asignan nombres de propiedades a valores de propiedad arbitrarios. De esta forma, se parecen más a las tablas *hash* o a las matrices asociativas (en *Perl*) que a las estructuras (en C) u objetos (en C ++ o Java). Su mecanismo de herencia OO está basado en prototipos como el del lenguaje poco conocido Self.

El núcleo del lenguaje *JavaScript* admite números, cadenas y valores booleanos como tipos de datos primitivos. También incluye soporte integrado para objetos de matriz, fecha y expresión regular. La versión integrada de este ejecuta *scripts* incrustados en páginas *web* HTML. Comúnmente se llama *client-side JavaScript* (del lado del cliente) para enfatizar que los *scripts* son ejecutados por la computadora cliente en lugar del servidor *web* (Flanagan, 2006).

PHP 7

PHP es un popular lenguaje multiplataforma y del lado del servidor integrado en HTML. En el nivel más básico, PHP puede hacer cualquier cosa que cualquier otro programa CGI (*Common Gateway Interface*) pueda hacer, como recopilar datos de formularios, generar contenido de páginas dinámicas o enviar y recibir

cookies. PHP se distribuye gratis bajo la licencia GNU y se usa en decenas de miles de sitios *web* de *internet* en todo el mundo (Bakken y otros, 2000).

UML 10.1

El lenguaje UML (Siglas en inglés de *Unified Modeling Language* o Lenguaje Unificado de Modelado en español) es un estándar OMG® (Siglas en inglés de *Object Management Group* que es un consorcio internacional de *software* sin fines de lucro que establece estándares en el área de objetos de computación distribuidos (Unified, 2018)) diseñado para visualizar, especificar, construir y documentar *software* orientado a objetos. Este estandariza 9 tipos de diagramas para representar gráficamente un sistema desde distintos puntos de vista (Booch y otros, 1999).

1.9.3 Framework de desarrollo

Symfony 3.4.9

Symfony es un completo *framework* (marco de trabajo) diseñado para optimizar el desarrollo de las aplicaciones *web*. El mismo separa la lógica de negocio, la lógica de servidor y la presentación de la aplicación *web*. Proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación *web* compleja. Además, automatiza las tareas más comunes, permitiendo al desarrollador dedicarse por completo a los aspectos específicos de cada aplicación. *Symfony* está desarrollado completamente con PHP.

Ha sido probado en numerosos proyectos reales y se utiliza en sitios *web* de comercio electrónico de primer nivel. *Symfony* es compatible con los gestores de bases de datos *MySQL*, *PostgreSQL*, *Oracle* Y *SQL Server* De *Microsoft*. Se puede ejecutar tanto en plataformas *nix (Unix, Linux, etc.) como en plataformas *Windows*.

Características de *Symfony*: (Potencier y Zaninotto, 2014)

- Fácil de instalar y configurar en la mayoría de plataformas.
- Independiente del sistema gestor de bases de datos.

- Mayormente sencillo de usar y a la vez adaptable a casos complejos.
- El desarrollador solo debe configurar aquello que no es convencional.
- Sigue la mayaría de las mejores prácticas y patrones de diseño para la *web*.
- Adaptable a las políticas y arquitecturas propias de cada empresa, y estable como para desarrollar a largo plazo.
- Código fácil de leer que incluye comentarios de phpDocumentor y que permite un mantenimiento muy sencillo.
- Fácil de extender, lo que permite su integración con librerías desarrolladas por terceros.

1.9.4 Entorno de desarrollo integrado

Un entorno de desarrollo integrado o *Integrated Development Environment* (IDE por sus siglas en inglés), es un programa informático compuesto por un conjunto de herramientas de programación. Un IDE 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.

NetBeans 8.2

Netbeans es un IDE que permite desarrollar de forma rápida y fácil aplicaciones de escritorio, móviles y *web* Java, así como aplicaciones HTML5 con HTML, *JavaScript* y *CSS*. El IDE también proporciona un gran conjunto de herramientas para desarrolladores PHP y C / C ++. Es gratuito y de código abierto y tiene una gran comunidad de usuarios y desarrolladores en todo el mundo (NetBeans, 2017).

1.9.5 Captura y almacenamiento de logs

Squid 3

Squid es un *proxy* de almacenamiento en caché para la *web* compatible con HTTP, HTTPS, FTP y más. Reduce el ancho de banda y mejora los tiempos de respuesta mediante el almacenamiento en caché y la reutilización de páginas *web* solicitadas con frecuencia. Posee amplios controles de acceso, además de funcionar como acelerador de servidores. Es multiplataforma y tiene licencia bajo GNU GPL.

Squid es ampliamente utilizado por proveedores de *internet* de todo el mundo para mejorar el acceso *web* de sus usuarios. Optimiza el flujo de datos entre el cliente y el servidor para mejorar el rendimiento y almacena en caché contenido de uso frecuente para ahorrar ancho de banda. También puede enrutar solicitudes de contenido a los servidores en una amplia variedad de formas para crear jerarquías de servidores de caché que optimicen el rendimiento de la red.

Miles de sitios *web* en *internet* usan *Squid* para aumentar drásticamente la entrega de contenido. Este puede reducir la carga del servidor y mejorar la velocidad de entrega a los clientes. Además, optimiza la entrega de contenido copiando solo lo referente a lo que se esté usando, en lugar de copiar todo de manera ineficiente. (Squid, 2017).

MySQL 5.5.32

MySQL es un sistema de gestión de bases de datos (SGBD) multiusuario, multiplataforma y de código abierto. Pertenece a la compañía sueca *MySQL AB*, a la que le pertenece casi todos los derechos del código fuente. Este gestor de base de datos es muy popular en aplicaciones *web*, y es componente de las plataformas LAMP, MAMP, WAMP, entre otras. Algunas de sus características son (Alvis Bettin, 2018):

- Está escrito en C y C++.

- Emplea el lenguaje SQL para consultas a la base de datos.

- *MySQL Server* está disponible como *freeware* bajo licencia GPL.

- *MySQL Enterprise* es la versión por suscripción para empresas, con soporte las

24 horas.

Elasticsearch 6.2.3

Elasticsearch es un motor de búsqueda basado en una API (en español, Interfaz de programación de aplicaciones) de código abierto para recuperación de información llamada Lucene. Provee un motor de búsqueda de texto completo (*full-text*) a través de una interfaz *web* RESTful. Básicamente ofrece un

servicio de búsquedas a través de una API RESTful. Mediante peticiones HTTP se almacena información de forma estructurada en Elasticsearch para que éste la indexe, y posteriormente poder realizar búsquedas sobre ella (Developerlove, 2017). Se usará *Squid* para capturar los *log* de los usuarios y poder almacenarlos en *Elasticsearch*.

1.9.6 Servidor web

Apache 2.4.18

El servidor *web Apache* es un *software* libre que funciona como contenedor donde se aloja la información. Es un servidor flexible, rápido y eficiente, de código abierto, continuamente actualizado y adaptado a los nuevos protocolos. Es multiplataforma de entre las que se encuentran: *FreeBSD*, *NetBSD*, *OpenBSD*, *GNU/Linux*, *Mac OS* y *Mac OS X Server*. Con los diferentes módulos de apoyo que proporciona y con la API de programación de módulos, puede ser adaptado a diferentes entornos y necesidades. Gracias a que es modular se han desarrollado diversas extensiones entre las que destaca PHP, un lenguaje de programación del lado del servidor (Sánchez y Echeverry, 2004).

1.10 Herramienta CASE

Visual Paradigm 10.1.0.0

Visual Paradigm es una potente herramienta CASE para visualizar y diseñar elementos de *software* que utiliza el lenguaje UML, proporciona una plataforma que les permite a los desarrolladores diseñar de forma rápida un producto con calidad. Es interoperable con otras herramientas CASE y se integra con los siguientes *software* Java: Eclipse/IBM *WebSphere*, *Jbuilder*, *NetBeans* IDE, *Oracle Jdeveloper*, BEA *Weblogic*. Está disponible en varias ediciones: *Enterprise*, *Professional*, *Community*, *Standard*, *Modeler* y *Personal*.

Se decidió utilizar el *Visual Paradigm* para visualizar y diseñar los elementos de *software*, debido a que utiliza el Lenguaje Unificado de Modelado (UML, por sus siglas en inglés). Tiene disponibilidad para disímiles versiones y para integrarse en múltiples plataformas. Esta herramienta necesita de altos requerimientos

computacionales para su óptima ejecución. Permite que se genere código en varios lenguajes (Saavedra López y otros, 2013).

Herramientas de prueba

JMeter 3.0

Es una aplicación de *software* de código abierto, una aplicación Java puro 100% diseñado para cargar la conducta funcional de prueba y medida de rendimiento. Fue diseñado originalmente para aplicaciones *web* de prueba, pero desde entonces se ha expandido a otras funciones de prueba. *JMeter* Se puede utilizar para probar el rendimiento tanto en recursos estáticos y dinámicos, aplicaciones dinámicas *web*. Con esta herramienta es posible simular una carga pesada en un servidor, grupo de servidores, la red o el objeto a probar su resistencia o para analizar el rendimiento general bajo diferentes tipos de carga (Mogollón, 2017).

Conclusiones del capítulo

El análisis de los principales elementos teóricos relacionados con el tipo de componente a desarrollar, así como la importancia de estos para la recuperación de información aportó los elementos necesarios para la comprensión del trabajo y el arribo a la solución deseada. El estudio de los sistemas homólogos demostró que de las aplicaciones existentes analizadas ninguna representa una posible solución y, aunque su estudio arrojó elementos a tener en cuenta para la implementación de la solución propuesta, quedó evidenciada la necesidad de desarrollar un componente para la visualización de trazas para el sistema Xilema Smart Keeper 3.0. El estudio de las metodologías, lenguajes y herramientas para el desarrollo permitió seleccionar un ambiente de desarrollo idóneo para implementar la solución propuesta.

Capítulo 2: Análisis y diseño de la propuesta de solución Componente de Visualización de trazas para el Sistema Xilema Smart Keeper 3.0

2.1 Introducción

En el presente capítulo se definen los requerimientos de *software*, las estructuras de datos y los artefactos necesarios para la solución de la propuesta del Componente de Visualización de trazas para el Sistema Xilema Smart Keeper 3.0. Se describen los principales requisitos funcionales y no funcionales de los que constará el componente. Son presentados la arquitectura, los patrones de diseño, los diagramas de clases y de secuencia y, los modelos de datos y de despliegue utilizados para la construcción de la propuesta de solución.

2.2 Características de la propuesta de solución

Para dar solución al problema planteado se propone desarrollar un componente de visualización de trazas del consumo de cuotas para el sistema Xilema Smart Keeper 3.0. Este sistema empleará *Squid* para capturar los *log* de los usuarios, los almacenará en *Elasticsearch* y con los datos capturados permitirá que los usuarios vean el historial de sus trazas de navegación y que puedan filtrar las mismas por dirección IP, dominio de la página visitada y por fechas específicas. Además, permitirá que los usuarios puedan consultar su cuota asignada y lo que han consumido de esta.

2.3 Requerimientos del sistema

Con el objetivo de satisfacer las necesidades de los clientes se especifican un conjunto de requisitos funcionales y no funcionales los cuales describen las especificidades del producto final.

“Los requerimientos funcionales describen lo que debe hacer el sistema de gestión de documentos; qué debe hacer, pero no cómo llevarlo a cabo” (Gómez-Domínguez y otros, 2003).

2.3.1 Requisitos funcionales

A continuación, se listan los requisitos funcionales de la propuesta de solución:

Tabla 1. Requisitos Funcionales del Componente de Visualización de trazas para el sistema Xilema Smart Keeper 3.0.

Requisito funcional	Descripción	Prioridad
Visualizar trazas de navegación.	El sistema debe mostrar los datos de dominio fecha e ip desde el que se accedió de las trazas de navegación del usuario autentificado.	Alta
Visualizar consumo de cuota.	El sistema debe mostrar la cantidad y porciento usados de la cuota de navegación asignada al usuario autentificado.	Alta
Aplicar filtros.	Permite al usuario filtrar la información de navegación por dominios, IP y fecha.	Alta

2.3.2 Requisitos no funcionales

Los requisitos no funcionales juegan un papel crítico durante el desarrollo del sistema, sirviendo como criterios de selección para elegir entre miríadas de diseños alternativos e implementaciones finales. En general, se reconoce que los errores de omisión o comisión al establecer y tener debidamente en cuenta tales requisitos son los más costosos y difíciles de corregir una vez que se ha implementado un sistema de *software* (Chung y otros, 2012).

RnF 1 Requisito de Interfaz de usuario 1. El sistema deberá tener una estructura clara, ordenando el contenido y las funciones de la aplicación en pestañas o apartados que abarquen todas las funcionalidades disponibles.

RnF 2 Requisito de Usabilidad 1. El componente debe tener un diseño orientado a una interfaz simple, visible, con una tipografía clara, intuitiva e interactiva, amigable para cualquier tipo de usuario.

RnF 3 Requisito de Usabilidad 2. El sistema debe proporcionar mensajes de error que sean informativos y orientados al usuario final.

RnF 4 Requisito de Hardware 1. El ordenador donde se ejecute el componente debe tener una memoria *RAM* de 2GB o superior y un procesador a una velocidad 2.10 GHz o superior.

RnF 5 Requisito de Software 1. Se requiere de un servidor de base de datos para el almacenamiento de *logs* (Elasticsearch 6.2.3).
RnF 6 Requisito de Software 2. El ordenador donde se instale el componente debe contar con un servidor proxy (*Squid3*).
RnF 7 Requisito de Software 3. Se requiere de un servidor de base de datos para el almacenamiento de los datos introducidos (*MySQL*).
RnF 8 Requisito de Software 4. El ordenador donde se ejecute el componente debe tener instalada la distribución de *GNU/Linux Ubuntu* 14.04 o superior.

RnF 9 Requisito de Seguridad 1. La información manejada por el componente debe estar protegida de accesos no autorizados, utilizando los mecanismos de autenticación del sistema Xilema Smart Keeper 3.0 para el control de acceso.

RnF 10 Requisito de Rendimiento 1. El sistema deberá responder en un máximo de 20 segundos a las peticiones del usuario.

RnF 12 Requisito de Mantenibilidad 1. Se debe estructurar el código de una manera consistente y predecible.

RnF 13 Requisito de Mantenibilidad 2. El diseño de las interfaces debe contemplar que las propiedades públicas y los parámetros de los métodos sean de un tipo común (estandarizados).

2.4 Historias de usuario

El cuarto escenario para la disciplina de requisitos de la metodología AUP-UCI, emplea las Historias de Usuario (HU) como forma de encapsular los requisitos. En estas se presenta la información suficiente para poder producir una estimación razonable del esfuerzo que tomará desarrollar una funcionalidad. A continuación, se muestran las HU que fueron generadas de los requisitos funcionales del sistema.

Tabla 2. HU_1 Visualizar trazas de navegación.

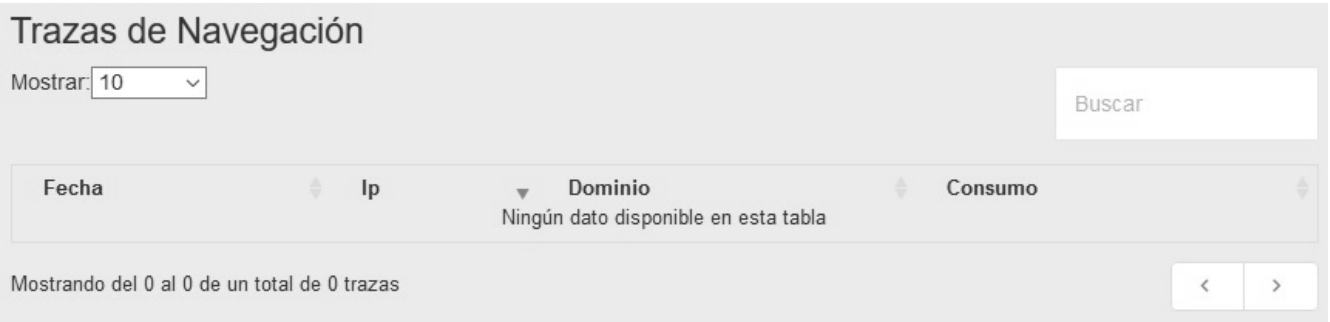
Historia de Usuario	
Número: HU_1	**Nombre:** Visualizar trazas de navegación
Programador responsable: Luis G. Fernández-Vega Rojas	**Iteración asignada:** 1
Prioridad: Alta	
Tiempo estimado: 12 horas	**Tiempo real:** 12 horas
Descripción: Permite que los usuarios visualicen las trazas de su navegación. Se debe ingresar un rango con una fecha inicial y una fecha final. Fecha inicial: Campo tipo data, obligatorio, en formato dd/mm/aa. Fecha final: Campo tipo data, obligatorio, en formato dd/mm/aa.	
Observaciones: Es necesario que el usuario ingrese un rango de fechas para poder visualizar las trazas pertenecientes al mismo. En el caso de que el usuario deje algún campo en blanco el sistema mostrará un mensaje de error donde se le comunica al usuario que no puede dejar el/los campo(s) en blanco.	
Prototipo de Interfaz de Usuario: 	

Tabla 3. HU_2 Visualizar consumo de cuota.

Historia de Usuario	
Número: HU_2	**Nombre:** Visualizar consumo de cuota
Programador responsable: Luis G. Fernández-Vega Rojas	**Iteración asignada:** 1

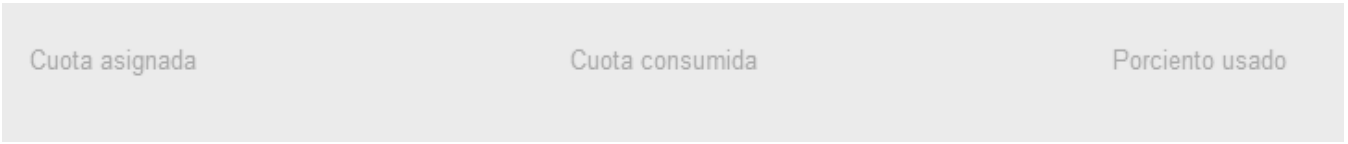
Prioridad: Alta	
Tiempo estimado: 8 horas	**Tiempo real:** 8 horas
Descripción: Permite que los usuarios visualicen el consumo de su cuota asignada.	
Observaciones: Es necesario que el usuario esté autentificado en el sistema para que se muestren los datos referentes al consumo de su cuota asignada.	
Prototipo de Interfaz de Usuario: 	

Tabla 4. HU_3 Aplicar filtros.

Historia de Usuario	
Número: HU_3	**Nombre:** Aplicar filtros
Programador responsable: Luis G. Fernández-Vega Rojas	**Iteración asignada:** 1
Prioridad: Media	
Tiempo estimado: 12 horas	**Tiempo real:** 12 horas
Descripción: Permite a los usuarios aplicar filtros a sus trazas de navegación para tener una información más detallada. Se puede filtrar por los siguientes parámetros: Dominio: Campo de texto. IP: Campo de texto. Fecha inicial: Campo tipo data, obligatorio, en formato dd/mm/aa. Fecha final: Campo tipo data, obligatorio, en formato dd/mm/aa.	

Observaciones: Una vez seleccionado un filtro se mostrarán las trazas agrupadas de acuerdo al parámetro solicitado.
Prototipo de Interfaz de Usuario: 

2.6 Patrón arquitectónico

La arquitectura de *software* demuestra la organización, funcionamiento y conexión entre las partes de un *software*. Su objetivo principal es garantizar un mejor desempeño en el desarrollo de las aplicaciones. Proporciona robustez, portabilidad y flexibilidad a la aplicación. Es considerado el elemento de enlace entre los requerimientos y la implementación del sistema (Gamma, 2013).

Para el desarrollo de la solución propuesta se decide utilizar el patrón arquitectónico Modelo-Vista-Controlador (en lo sucesivo MVC), dado que el marco de trabajo *Symfony* 3.4.9 está diseñado sobre en el mismo.

El patrón Modelo-Vista-Controlador separa los datos, la interfaz de usuario y la lógica de control de una aplicación en tres componentes distintos, estos son:

Modelo: Encapsula los datos y las funcionalidades. Es independiente de cualquier representación de salida y comportamiento de entrada. El modelo está representado por los archivos existentes en los directorios *SearchRepository, Model* y *Entity* como son LogRepository.php, LogModel.php y Log.php respectivamente.

Vista: Muestra la información al usuario. Pueden existir múltiples vistas del modelo. Cada vista tiene asociado un componente controlador. Las vistas del componente se encuentran agrupadas en el directorio *Resources/views*, y son los archivos con extensión .html.twig.

Controlador: Reciben las entradas o eventos que codifican los movimientos o pulsación de botones del ratón o teclas. Los eventos son traducidos a solicitudes de servicio para el modelo o la vista. Las clases controladoras utilizadas en el componente son TrazasController.php y AutentificationController.php y se encuentran en el directorio *Controller*.

Algunas de las ventajas del empleo de este patrón arquitectónico son (García, 2011):

- Facilitar la agregación de múltiples representaciones de los mismos datos.
- Manejar muchas peticiones con el mismo rendimiento simplemente añadiendo más *hardware*; esto representa una alta escalabilidad.
- Facilitar el mantenimiento de errores.
- Crear independencia de funcionamiento.

2.7 Patrones de diseño

Los patrones de diseño permiten la reutilización de diseños exitosos, empleando experiencias previas para nuevos diseños. Estos patrones resuelven problemas específicos de diseño, y vuelven el diseño orientado a objetos más flexible, elegante y extremadamente reutilizable. Son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de *software* y otros ámbitos del diseño de interacción o interfaces (Pressman, 2011).

Se clasifican en dos categorías: GRASP (Siglas en inglés de *General Responsibility Assignment Software Patterns* o Patrones Generales de *Software* para Asignación de Responsabilidades en español) y GoF (Siglas en inglés de *Gang-Of-Four* o La Pandilla De Los Cuatro en español) (Pressman, 2011). Para el diseño del Componente de Visualización de trazas para el sistema Xilema Smart Keeper 3.0 se tuvieron en cuenta los siguientes patrones:

2.7.1 Patrones Generales de Software para la Asignación de Responsabilidades (GRASP)

Los GRASP son patrones generales de *software* para asignación de responsabilidades a objetos. Resulta en un apoyo para la enseñanza que ayuda a entender el diseño de objeto esencial y aplica el razonamiento para el diseño de una forma sistemática, racional y aplicable (Visconti y Astudillo, 2011).

Alta Cohesión: Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas, que colaboran entre sí y con otros objetos para simplificar su trabajo. Las mismas poseen un número relativamente pequeño de responsabilidades, definiendo así que cada clase realice solo las funcionalidades para las cuales fueron creadas, generando un bajo acoplamiento y fomentando la reutilización (Visconti y Astudillo, 2011).

El Framework Symfony tiene como característica la organización del trabajo en cuanto a la estructura del proyecto, lo cual permite crear y trabajar con clases con una alta cohesión. Esto se puede observar en el sistema en donde cada clase controladora se ajusta a manejar solo las responsabilidades correspondientes a las entidades con las que se relaciona. Esto hace posible que se puedan realizar grandes cambios en el sistema con repercusiones mínimas.

Bajo Acoplamiento: El uso de este patrón garantiza que las clases estén lo menos ligadas posible entre sí, de tal forma que, en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de las clases, potenciando la reutilización y disminuyendo la dependencia entre las clases (Larman, 2003). En el sistema las clases controladoras no se relacionan entre sí, lo que disminuye las dependencias entre las mismas.

Experto: Se utiliza en la asignación de responsabilidades y en el diseño de objetos. Expresa que se deben asignar las responsabilidades a aquellos objetos que disponen de la información para hacerlo. Esto provoca el encapsulamiento de la información. Generalmente, esto conlleva un bajo acoplamiento, dando lugar a sistemas más robustos y más fáciles de mantener (Larman, 2003). En el componente se evidencia el uso de este patrón al conectar las vistas al controlador. Los archivos **routing.yml** y **trazas.yml** contienen el mapa de rutas URL que se enlazan a las acciones de los controladores del componente.

2.7.2 Patrones Gang-Of-Four (GOF)

El catálogo de patrones más famoso es el contenido en el libro "*Design Patterns: Elements of Reusable Object-Oriented Software*", también conocido como: El libro GOF (Gang-Of-Four Book). Según este documento, estos patrones se clasifican según su propósito en creacionales, estructurales y de comportamiento, mientras que respecto a su ámbito se clasifican en clases y objetos:

Estructural:

Decorator (Decorador): Permite añadir responsabilidades adicionales a un objeto dinámicamente, proporcionando una alternativa flexible a la especialización mediante herencia, cuando se trata de añadir funcionalidades. En *Symfony* 3.4.9 la vista se separa en una plantilla base y varias plantillas que heredan de esta. Normalmente, la plantilla base es global en toda la aplicación y contiene el código HTML que es común a la mayoría de las páginas, lo cual es una implementación del patrón Decorador. Su uso aporta una mayor flexibilidad que la herencia estática, permitiendo, entre otras cosas, añadir una funcionalidad dos o más veces (Labrada y Aragón, 2013).

Comportamiento:

Command (Comando): Es un patrón de diseño de comportamiento de objetos. Permite manipular y encapsular las peticiones de los usuarios y enviarlas a un objeto encargado de darle respuesta. Su uso es apropiado cuando lo fundamental en la relación entre una petición y la acción que la satisface es la flexibilidad (Labrada y Aragón, 2013).

Template Method (Método Plantilla): Define una estructura algorítmica en la súper clase, delegando la implementación a las subclases. Es decir, define una serie de pasos, en donde los pasos serán redefinidos en las subclases (polimorfismo). Un ejemplo del uso de este patrón en *Symfony* 3.4.9 se ve reflejado en las plantillas *twig*. Estas permiten la herencia entre clases, así como la redefinición de los métodos a fin de lograr el diseño deseado (Labrada y Aragón, 2013).

2.8 Diagramas de clases del diseño

Un diagrama de clases de diseño muestra la especificación para las clases *software* de una aplicación. Entre la información que representa se incluyen (UNAD, 2016):

- Clases, asociaciones y atributos.
- Interfaces, con sus operaciones y constantes.
- Métodos.
- Navegabilidad.
- Dependencias

A continuación, se exponen los diagramas de clases de diseño correspondientes a las historias de usuario de la propuesta de solución.

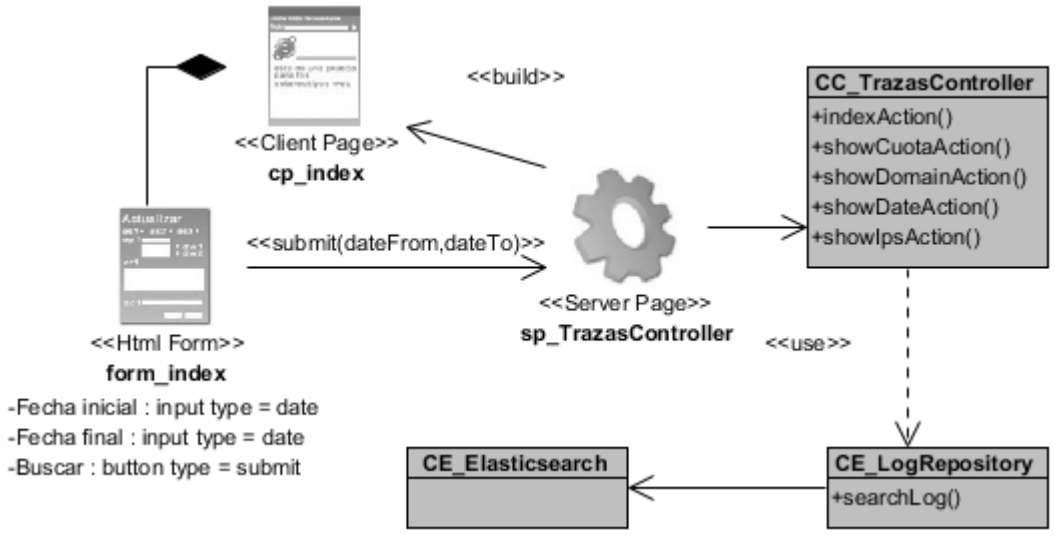


Figura 1. Diagrama de clases de diseño de la HU_1 Visualizar trazas de navegación.

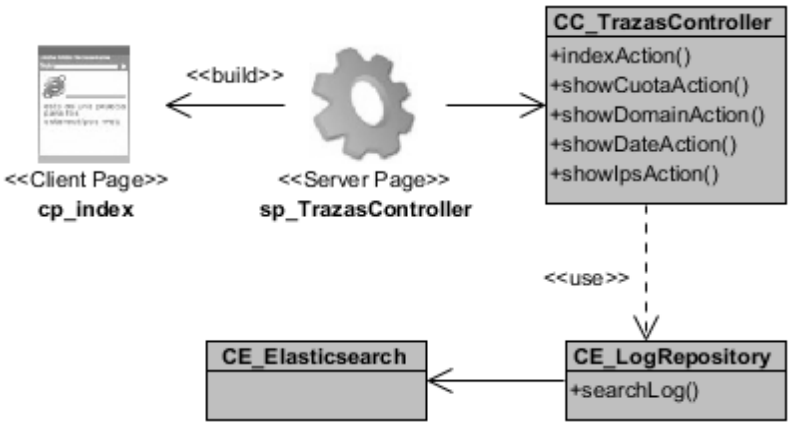


Figura 2. Diagrama de clases de diseño de la HU_2 Visualización del consumo de cuota.

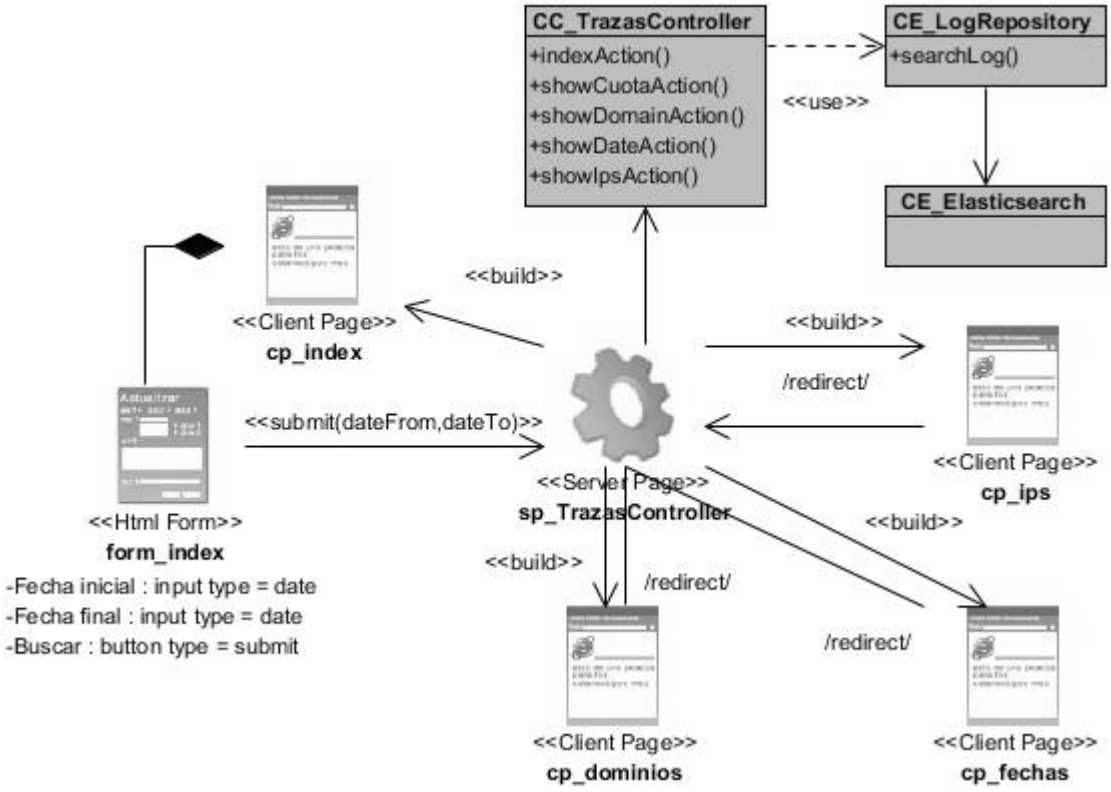


Figura 3. Diagrama de clases de diseño de la HU_3 Aplicar filtros.

Descripción de las clases de las HU_1, HU_2 y HU_3:

Clase<<cp_index>>

Página principal del componente de visualización de trazas. Contiene el formulario para ingresar el rango de fechas del cual se desea visualizar las trazas.

Clase<<cp_index>>

Página que contiene las trazas filtradas por dominios.

Clase<<cp_fechas>>

Página que contiene las trazas filtradas por fechas.

Clase<<cp_ips>>

Página que contiene las trazas filtradas por ips.

Clase<<sp_TrazasController>>

Contiene las consultas para procesar las peticiones del usuario al componente.

Clase<<LogRepository>>

Clase intermediaria que contiene las consultas especificas hechas al *Elasticsearch*.

Clase<<CE_Elasticsearch>>

Contiene los *log* indexados generados de la actividad del usuario.

2.9 Diagrama de Secuencia

Los diagramas de secuencia en el UML son utilizados principalmente para modelar las interacciones entre los actores y los objetos en un sistema, así como las interacciones entre los objetos en sí (Sommerville,

2011). A continuación, se muestran los diagramas de secuencia que se generaron en la presente investigación, en correspondencia con los requisitos funcionales establecidos en el epígrafe 2.3.1.

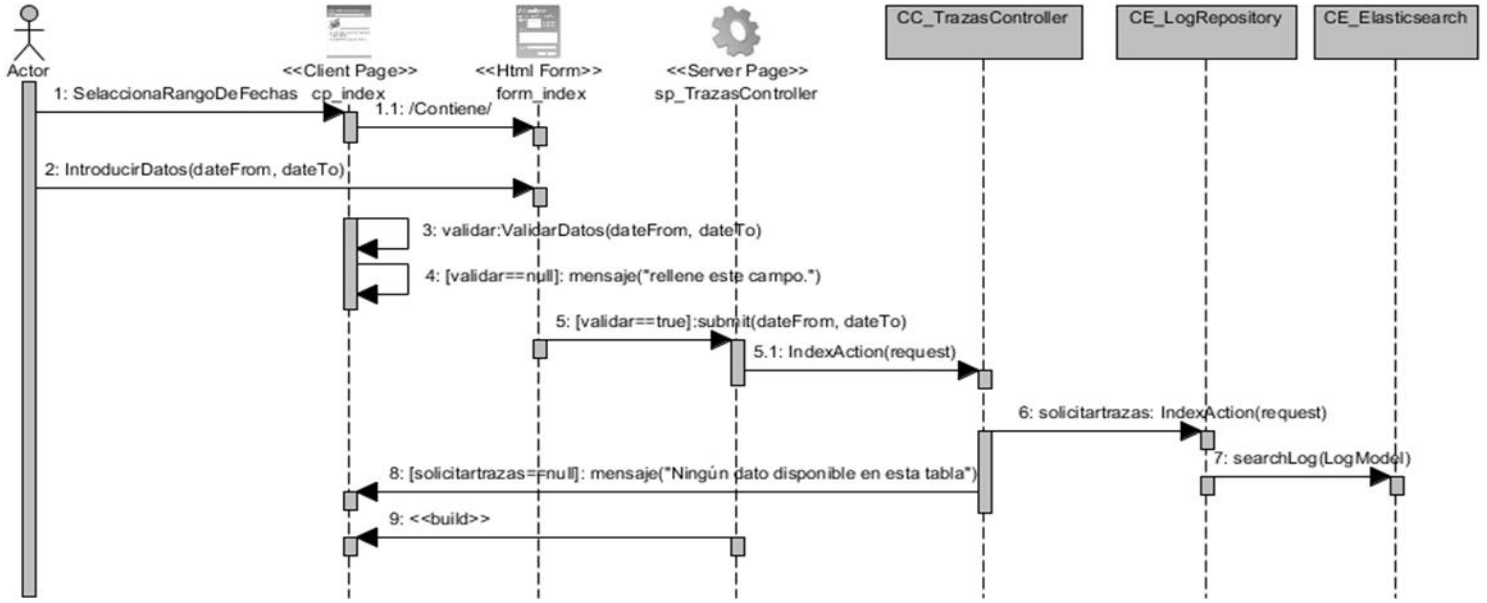


Figura 4. Diagrama de secuencia HU_1 Visualizar trazas de navegación.

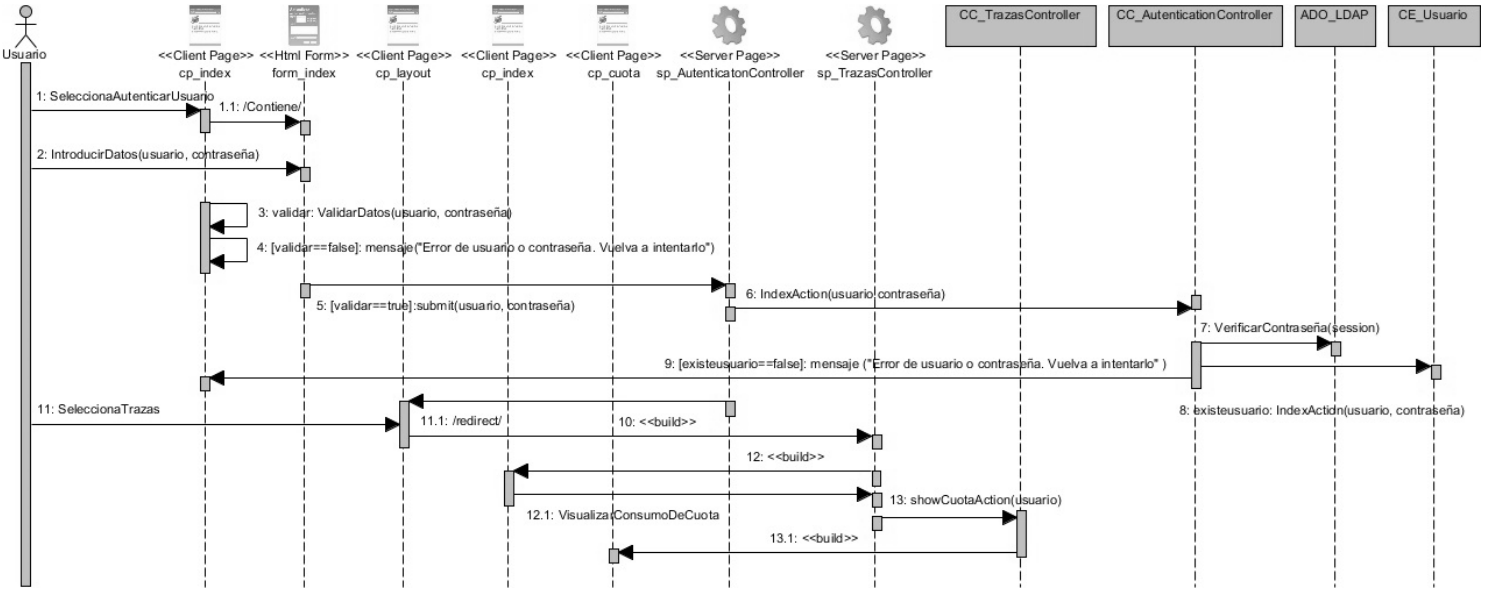


Figura 5. Diagrama de secuencia HU_2 Visualizar consumo de cuota.

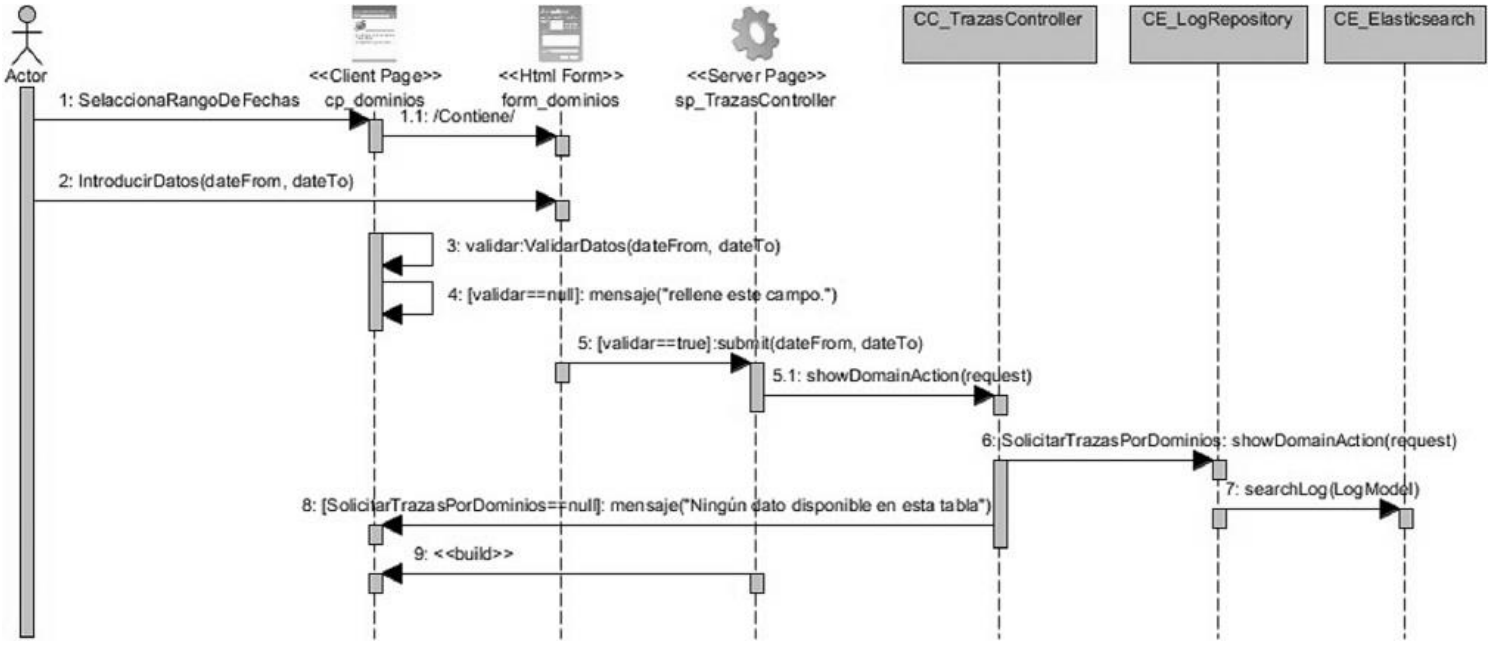


Figura 6. Diagrama de secuencia HU_3 Aplicar filtros.

2.10 Modelo de datos

Dada la importancia que representa la persistencia de los datos en la arquitectura de una aplicación informática se hace necesario la elaboración de un diseño de base de datos en correspondencia con las necesidades que debe cumplir el *software* a desarrollar. Esto tiene como función la representación clara y concisa de las clases entidades y la relación entre las mismas. En el caso de la propuesta de solución planteada es necesario el almacenamiento de la información capturada por *Squid* en formato de *logs*, la cual se genera de la actividad de navegación de los usuarios. A continuación, se detalla la función de las tablas que se encuentran en la base de datos.

usuario: Almacena la información de autentificación del usuario en el sistema.

***log*:** Almacena la información capturada de la actividad del usuario en la red la cual será mostrada por el componente desarrollado**.**

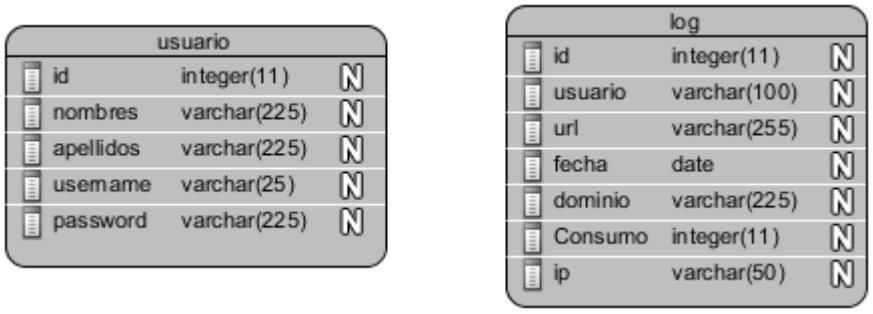


Figura 7. Modelo de datos del Componente de Visualización de trazas para el sistema Xilema Smart Keeper 3.0.

2.11 Modelo de despliegue

Para lograr un despliegue exitoso de la aplicación, se hace necesario mostrar la disposición física de los distintos nodos que componen el sistema, así como la distribución de los componentes en dichos nodos. El diagrama de despliegue presentado a continuación servirá para modelar la topología de hardware y la distribución del sistema necesarias.

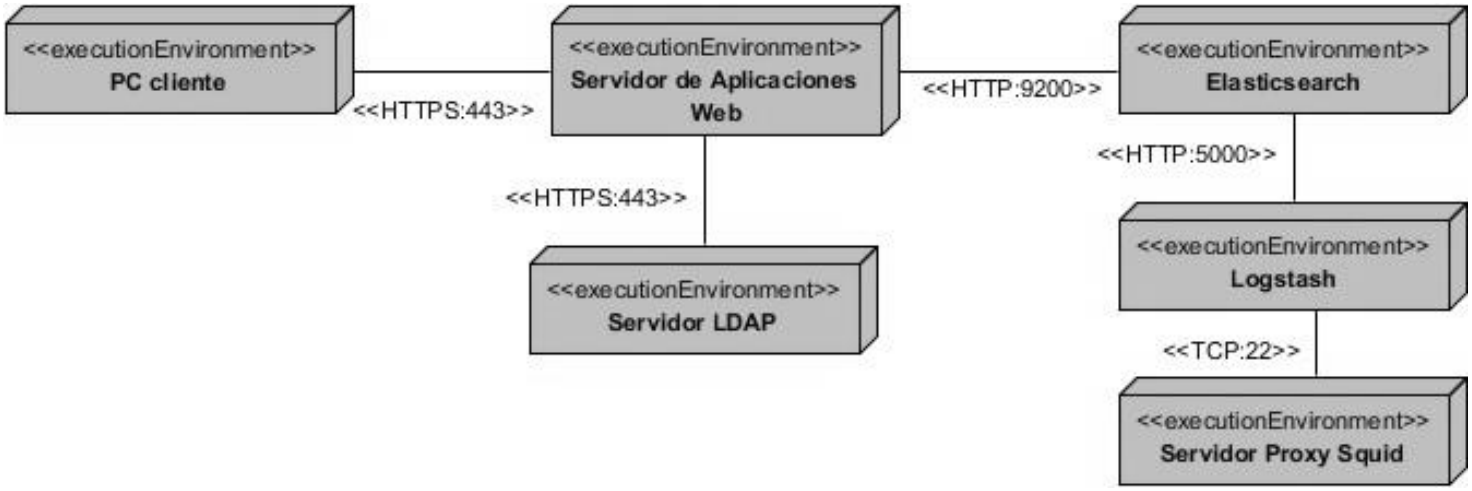


Figura 8. Diagrama de Despliegue del Componente de Visualización de trazas para el sistema Xilema Smart Keeper 3.0.

En este diagrama se definen el uso de estaciones de trabajo (Ambiente de ejecución) que el usuario utilizará para conectarse vía *HTTPS* (Siglas en inglés de *HyperText Transfer Protocol Secure* o Protocolo Seguro de

Transferencia de Hipertexto en español) con el servidor de aplicaciones *web* que hospeda al sistema Xilema Smart Keeper 3.0. También se puede apreciar la necesidad de conexión *HTTPS* con un servidor *LDAP* (Siglas en inglés de *Light Directory Access Protoco*l o Protocolo Ligero de Acceso a Directorios en español) que garantiza los datos necesarios para la autenticación en el sistema. Para las conexiones entre el servidor de la aplicación y Elasticsearch, así como la de este con *Logstash* se emplea *HTTP* y también se aprecia el uso del protocolo *TCP* (Protocolo de Control de Transmisión) para conectar este último con el servidor proxy Squid.

Conclusiones del capítulo

La realización de las historias de usuario ofrece una visión más clara de los requisitos funcionales impuestos para el Componente de Visualización de trazas para el sistema Smart Keeper 3.0. La arquitectura Modelo-Vista-Controlador seleccionada permitió definir la estructura del *software* y la interrelación entre sus diferentes componentes. EL diseño del conjunto de diagramas de clases y de secuencias facilitó una visión en cuanto a composición física y lógica del sistema. Por tanto, identificados los patrones de diseño y arquitectura, la solución propuesta cuenta con un alto grado de resistencia ante posibles modificaciones.

Capítulo 3: Implementación y validación del Componente de Visualización de trazas para el sistema Xilema Smart Keeper

3.1 Introducción

Es en este punto de la implementación, en el desarrollo de un producto de *software*, cuando todas las descripciones y arquitecturas propuestas en las fases de análisis y diseño son puestas en práctica, complementando todas las fases anteriores dentro del proceso de desarrollo de *software*. La implementación se convierte en la materialización de los requisitos siendo su objetivo la conformación del producto final requerido por el o los clientes.

En conjunto con proceso de implementación, es necesario que el *software* que se crea deba someterse a determinadas pruebas que garanticen que los requisitos definidos en las etapas anteriores correspondan con el producto final. Esta etapa es conocida como validación del sistema, y en ella es posible aplicar diferentes tipos de pruebas en correspondencia con lo que se desea verificar.

3.2 Diagrama de Componentes

Este tipo de diagrama ilustra la relación existente entre los componentes de *software* y su ubicación dentro del sistema, como están implementadas las clases en términos de componentes y las dependencias entre los mismos. Describe también como se organizan los componentes de acuerdo con los mecanismos de estructuración disponibles en el marco de trabajo y del lenguaje de programación utilizado.

Tabla 5. Descripción de los componentes del Diagrama de Componentes.

Componentes					Descripción
Xilema Smart Keeper 3.0	Componente de Visualización de trazas para el sistema Xilema Smart Keeper	Modelo	Model	LogModel.php	Encapsula los parámetros de los datos a emplear.
			Repository	LogRepository.php	Clase responsable de realizar las peticiones de los datos.
		Controlador	TrazasController.php		Clase controladora que contiene las funciones relacionadas con la visualización de las trazas, su filtración por dominios, fechas e ips, y la información sobre el consumo de la cuenta del usuario.
		Form	LogSearchType.php		Clase encargada de construir el formulario relacionado con la filtración de las trazas.
		Vista	trazas	index.html.twig	Muestra un listado de las trazas del usuario registrado dentro del rango de fechas ingresado, así como información sobre su consumo de cuenta.
				layout.html.twig	Permite aplicar una estructura de herencia de planillas twig.
				navbar.html.twig	Muestra la barra de navegación del componente.
				dominios.html.twig	Muestra un listado de las trazas del usuario filtradas por dominio.
				fechas.html.twig	Muestra un listado de las trazas del usuario filtradas por fechas.

				ips.html.twig	Muestra un listado de las trazas del usuario filtradas por IP.
				cuota.html.twig	Muestra el consumo de cuota del usuario registrado.
		routing.yml			Contiene el mapa de rutas URL que se enlazan a las acciones de los controladores del componente.
	Controlador	AutenficationController.php			Clase controladora responsable de la autentificación de los usuarios en el sistema.
	Vista	Login	Index.html.twig		Muestra la interfaz que permite la autentificación del usuario en el sistema.
	config.yml				Contiene las configuraciones generales del sistema Xilema Smart Keeper 3.0.

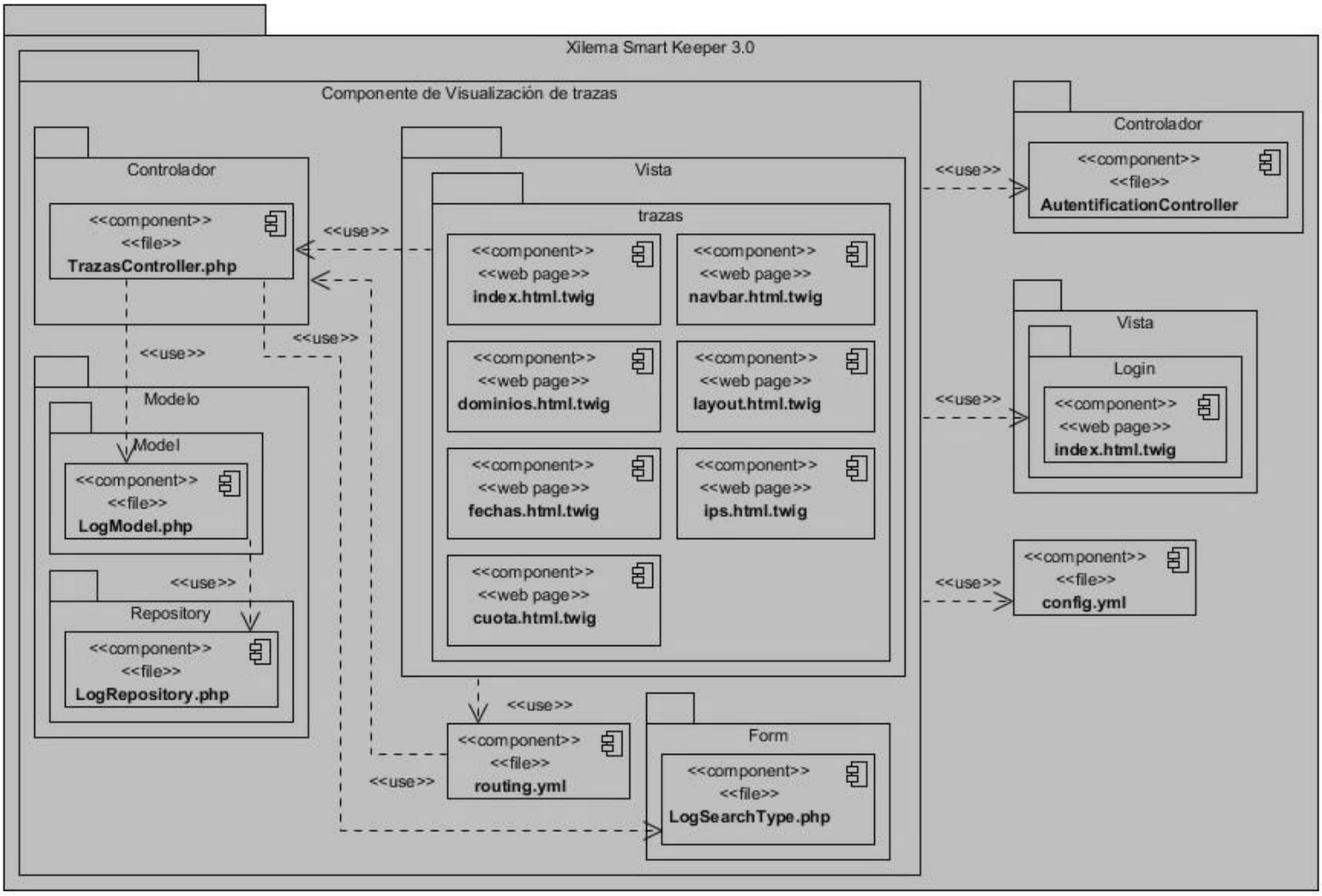


Figura 9. Diagrama de componentes.

3.3 Estándares de codificación

Los estándares de codificación indican cuales son las reglas mínimas para realizar algún proceso, elementos como declarar una variable, cuál debe ser el nombre de las clases o como se deben crear los paquetes se encuentran entre estos. Cabe destacar que los estándares de codificación en algunos casos se presentan como sugerencia, pero no pero no como imposiciones. Aparte de lo anteriormente mencionado algunas de las ventajas de aplicar los estándares de codificación son: (Henao, 2014)

- Facilitar la lectura y entendimiento del código fuente.
- Reducción del costo económico y el esfuerzo del mantenimiento del código.
- Por lo regular en todo *software* intervienen diferentes desarrolladores, si todos manejan el mismo lenguaje y convenciones, el desarrollo será mucho más ágil.
- Permitir manejar un estándar de programación en el equipo de trabajo.
- Facilitar la agregación o la modificación de funcionalidades.
- El seguir las convenciones y aplicar los estándares muestra prestigio como desarrollador.

Con el objetivo de facilitar el entendimiento del código y precisar un modelo a base, se ha seguido el estándar de codificación perteneciente a *Symfony* 3.4.9. Los mismos se muestran a continuación separados por los diferentes aspectos en los que se encuentran agrupados.

Estructura:

1. Añadir un solo espacio después de cada delimitador coma.
2. Añadir un solo espacio alrededor de los operadores (==, &&, ...).
3. Añadir una coma después de cada elemento del arreglo en un arreglo multilínea, incluso después del último.
4. Añadir una línea en blanco antes de las declaraciones *return*, a menos que el valor devuelto solo sea dentro de un grupo de declaraciones (tal como una declaración *if*).
5. Usar llaves para indicar la estructura del cuerpo de control, independientemente del número de declaraciones que contenga.
6. Definir una clase por archivo, esto no se aplica a las clases ayudantes privadas, de las cuales no se tiene la intención de crear una instancia desde el exterior.
7. Declarar las propiedades de clase antes que los métodos.
8. Primero declarar los métodos públicos, luego los protegidos y finalmente los privados.
9. Usar paréntesis al crear instancias de clases independientemente de la cantidad de argumentos que tenga el constructor.

Convenciones de nomenclatura:

1. Utilizar mayúsculas intercaladas —sin guiones bajos— en nombres de variable, función, método o argumentos.
2. Usar guiones bajos para nombres de opción y nombres de parámetro;
3. Utilizar espacios de nombres para todas las clases.

Documentación:

1. Añadir bloques PHPDoc a todas las clases, métodos y funciones.
2. Omitir la etiqueta *@return* si el método no devuelve nada.
3. Las anotaciones *@package* y *@subpackage* no se utilizan.

3.4 Validación del sistema

La validación del sistema se refiere a la actividad que busca asegurar que el *software* creado se ajuste a los requisitos requeridos. En el proceso de validación, las pruebas de *software* son complementarias para el análisis y comprobación del sistema. Estas implican ejecutar una implementación del *software* con datos de prueba, examinándose las salidas y entorno operacional de *software* para comprobar que funciona tal y como se requiere. Las pruebas de validación intentan demostrar que el *software* funciona correctamente y satisface sus requerimientos (Valles, 2014).

3.4.1 Pruebas de funcionalidad

Las pruebas funcionales son un proceso de control de calidad que consiste en asegurar el cumplimiento de un sistema o componente con requerimientos funcionales. Estas pruebas pueden realizarse durante la fase de desarrollo, individualmente para secciones específicas desarrolladas por el equipo de trabajo, al final del desarrollo del proyecto o después de la integración de los diferentes componentes del sistema. El objetivo principal de las pruebas funcionales es analizar el producto terminado y determinar si realiza correctamente las funciones con las que debe cumplir (Crowdsourcedtesting, 2018).

Las pruebas de funcionalidad aplicadas al presente proyecto se muestran a continuación:

Caso de prueba de la HU_1 Visualizar trazas de navegación

Esta prueba verifica que la función del componente que permite la visualización de las trazas de navegación opere de manera correcta.

Tabla 6. Caso de Prueba del RF1_Visualizar trazas de navegación.

Escenario	**Descripción**	**Respuesta del sistema**	**Flujo Central**
EC1.1 Se ingresa un rango de fechas válido para visualizar las trazas de navegación.	El usuario ingresa un rango de fecha en el sistema y se comprueba que sea válido. De ser válido, se muestra una interfaz con las trazas de su navegación durante el intervalo seleccionado.	El sistema verifica si las fechas ingresadas son válidas. De ser válidas, muestra una interfaz con las trazas del usuario ordenadas por fecha, dominio e ip.	1 - Se ingresa el rango de fechas del que se desean visualizar las trazas de navegación. 2 - Se selecciona la opción Buscar. 3 - El sistema muestra las trazas de navegación del usuario.
EC1.2 Se solicita la información sin ingresar un rango de fechas.	El usuario solicita la información al sistema sin ingresar un rango de fechas. Se muestra un mensaje para indicarle al usuario que debe ingresa una fecha.	El sistema comprueba que se haya ingresado un rango de fecha. De encontrarse alguno de los campos en blanco, se muestra el mensaje: “Rellene este campo”.	1 - Se selecciona la opción buscar sin haber ingresado un rango de fecha. 2 - El sistema muestra un mensaje de error.

Caso de prueba de la HU_2 Visualizar consumo de cuotas

Esta prueba verifica que la función del componente que permite la visualización del consumo de cuota opere de manera correcta.

Tabla 7. Caso de Prueba del RF2_Visualizar consumo de cuotas.

Escenario	**Descripción**	**Respuesta del sistema**	**Flujo Central**

EC2.1 Visualizar consumo de cuota.	Una vez que el usuario autentificado seleccione la opción Trazas en la interfaz principal del sistema se muestra una interfaz con la información referente a su cuota.	El sistema muestra una interfaz con la información de la cuota del usuario autentificado.	1 - Se autentica el usuario en el sistema. 2 - Se selecciona la opción Trazas. 3 - El sistema muestra una interfaz con los datos de la cuota del usuario.

Caso de prueba de la HU_3 Aplicar filtros

Esta prueba verifica que la función del componente que permite la aplicación de filtros a las trazas de navegación opere de manera correcta.

Tabla 8. Caso de Prueba del RF2_Aplicar filtros.

Escenario	**Descripción**	**Respuesta del sistema**	**Flujo Central**
EC3.1 Se ingresa un rango de fechas válido para visualizar las trazas de navegación filtradas.	El usuario ingresa un rango de fecha en el sistema y se comprueba que sea válido. De ser válido, se muestra una interfaz con las trazas de su navegación durante el intervalo seleccionado filtradas por el parámetro seleccionado.	El sistema verifica si las fechas ingresadas son válidas. De ser válidas, muestra una interfaz con las trazas del usuario ordenadas por fecha, dominio e ip.	1 - Se ingresa el rango de fechas del que se desean visualizar las trazas de navegación filtradas por el parámetro seleccionado. 2 - Se selecciona la opción Buscar. 3 - El sistema muestra las trazas de navegación del usuario filtradas por el parámetro seleccionado.

EC1.2 Se solicita la información sin ingresar un rango de fechas.	El usuario solicita la información al sistema sin ingresar un rango de fechas. Se muestra un mensaje para indicarle al usuario que debe ingresa una fecha.	El sistema comprueba que se haya ingresado un rango de fecha. De encontrarse alguno de los campos en blanco, se muestra el mensaje: “Rellene este campo”.	1 - Se selecciona la opción buscar sin haber ingresado un rango de fecha. 2 - El sistema muestra un mensaje de error.

Resultado de las pruebas de funcionalidad

En una primera iteración se detectaron un total de 7 no conformidades, 5 de ellas de funcionalidad y 2 de validación. Todas las no conformidades fueron corregidas con éxito. En una segunda iteración se identificaron 4 nuevas no conformidades de funcionalidad, las cuales también fueron corregidas. Finalmente, para una tercera iteración no se detectó ninguna no conformidad obteniéndose resultados satisfactorios. En la siguiente gráfica (Figura 10) se pueden apreciar los resultados alcanzados:

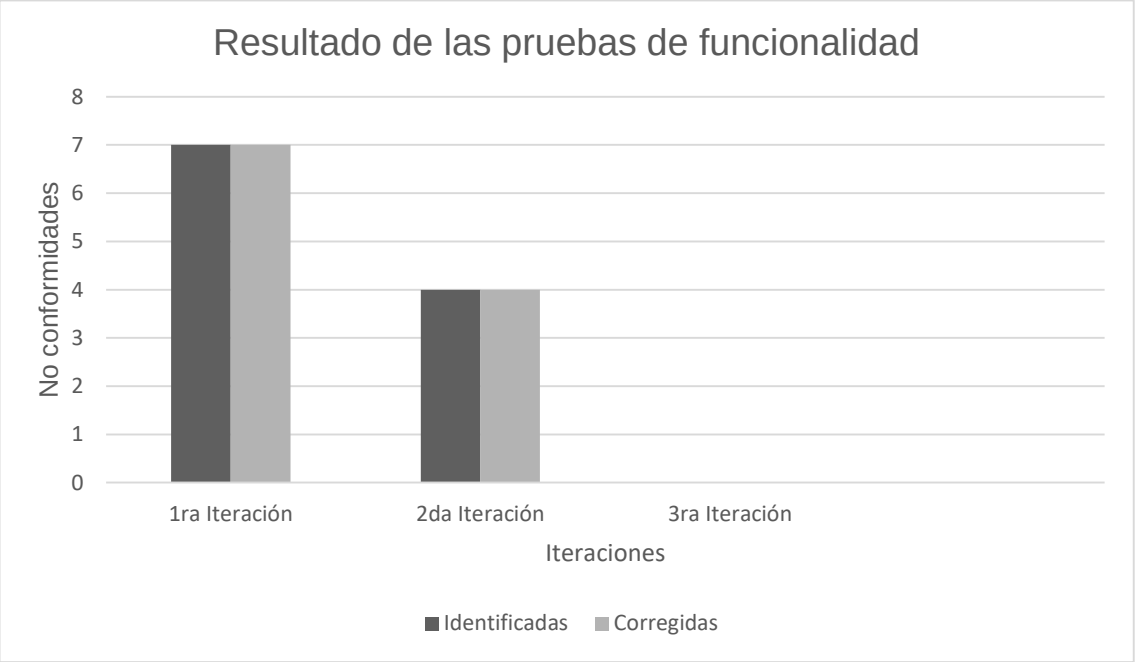


Figura 10. Resultado de las pruebas funcionales.

3.4.2 Pruebas de carga y estrés

Las pruebas de carga y estrés permiten la detección de problemas de rendimiento en las aplicaciones, evalúan si se satisfacen las necesidades del cliente antes de su publicación, y determinan las respuestas de la aplicación ante distintos niveles de uso a los que puede verse sometida una vez publicada. En este tipo de pruebas se simula el acceso al sistema de varios usuarios de manera simultánea, siguiendo el patrón de prueba que se haya creado (ITblogsogeti, 2017).

Teniendo en cuenta las posibles acciones que puede realizar el usuario en el sistema se diseñó un plan de pruebas de rendimiento para el componente. Con una muestra de 1400 peticiones concurrentes con un periodo de subida de un segundo la herramienta *JMeter* generó el reporte mostrado en la tabla 9.

Tabla 9. Resultados de las pruebas de rendimiento mediante el uso de JMeter.

Petición	**Cantidad de peticiones**	**Tiempo de respuesta (ms)**	**Tasa de error (%)**
Visualizar trazas de navegación.	1400	24	0
Visualizar consumo de cuota.	1400	15	0
Visualizar filtrar trazas por dominios.	1400	20	0
Visualizar filtrar trazas por fechas.	1400	20	0
Visualizar filtrar trazas por ips.	1400	20	0
Total	6000	19.8	0

Como se muestra en la tabla 9, de acuerdo al análisis del reporte generado por la herramienta *JMeter*, para un total de 6000 muestras realizadas al componente, se alcanzó un rendimiento de 19.8 peticiones por segundo, con 0% de porcentaje de errores para cada petición realizada. Partiendo de estos resultados queda determinado que el Componente de Visualización de trazas para el sistema Xilema Smart Keeper 3.0 responde de manera correcta ante situaciones de carga y estrés en función de la muestra empleada.

3.4.3 Pruebas de integración

Las pruebas de integración parten de los componentes individuales previamente probados y tienen como objetivo descubrir errores que se pueden producir en la interacción entre estos. Las interfaces entre los componentes suelen ser fuente de errores de integración, pero también los tratamientos concurrentes, como el acceso a estructuras de datos comunes o a la definición de transacciones sobre las bases de datos (Sommerville, 2005).

Con el objeto de validar la compatibilidad y el correcto funcionamiento de las interfaces que comunican los diferentes componentes que componen el sistema se hace necesario la realización de pruebas de integración.

Para la realización de estas pruebas se ejecutaron las siguientes tareas:

1- Comprobación del funcionamiento del enlace entre el Sistema Xilema Smart Keeper 3.0 y el Componente de Visualización de trazas.
2- Verificación de la conexión entre la autentificación en el Sistema Xilema Smart Keeper y el Componente de Visualización de trazas.
3- Verificación de la conexión entre el Componente de Visualización de trazas, el motor de búsqueda *Elasticsearch* y el Servidor Proxy Squid respectivamente.
4- Validación de las rutas de acceso, en el *routing* principal del Sistema Xilema Smart Keeper 3.0.

Conclusiones del capítulo

La elaboración del diagrama de componentes, permitió una mejor comprensión de la estructura de los archivos principales del componente implementado. La estandarización del código facilitó su comprensión y orden durante la implementación. Con la aplicación de las distintas pruebas realizadas en este capítulo, se ha evidenciado el correcto funcionamiento del complemento desarrollado y se ha corroborado que los requisitos funcionales y no funcionales propuestos para dar solución a la problemática planteada han sido satisfechos.

Conclusiones generales

Sobre la presente investigación se puede concluir lo siguiente:

1. El estudio del estado del arte de los componentes de visualización de trazas permitió demostrar que los sistemas existentes a nivel nacional e internacional no cumplen con las necesidades requeridas para su implementación en el sistema deseado.

2. El seguimiento de la metodología de desarrollo y el levantamiento de requisitos permitieron una mayor comprensión del componente a desarrollar, la identificación de sus principales procesos, características y posibilitó crear un diseño del componente que se ajustara a las necesidades planteadas.

3. El Componente de Visualización de trazas permite a los usuarios ver la información referente al consumo de su cuota y de su navegación por la red, así como poder filtrar esta última por parámetros como dominio, fecha e ip.

4. Las pruebas realizadas garantizan que el componente implementado funcione correctamente y que se integre adecuadamente con el resto de los componentes del Sistema Xilema Smart Keeper 3.0.

Bibliografía referenciada

IV CONGRESO de la CiberSociedad 2009 Crisis Analógica, futuro digital [en línea]. [Fecha de consulta: 23 noviembre 2017]. Disponible en: http://www.cibersociedad.net/congres2009/po/coms/elementos-de-la-visualizacion-de-datos-y-redes/972/

AGUILERA, Yoel Benedico y GONZÁLEZ, Yankier Crespo. SISTEMA PARA EL CONTROL DE TRAZAS DE UN SERVIDOR PROXY (SCTRAZAS). Universidad&Ciencia, 2017, vol. 6, no 2, p. 1-16.

ALVIS BETTIN, Alvaro Simón; VEGA VÁSQUEZ, Miller Haseen. Desarrollo de un software para el registro de visitas a pacientes en el hospital del Municipio de Sahagún, haciendo uso de sistemas biométricos. Tesis Doctoral. 2018. p. 29.

BAKKEN, Stig Saether; SURASKI, Zeev y SCHMID, Egon. PHP Manual: Volumen 2. iUniverse, Incorporated, 2000.

BALLARI, Tulio L. Una Implementación de Proxy Interceptor con Linux Kernel 2.4 y Squid 2.3 Stable 4. 2002.

BOOCH, G., et al. El lenguaje unificado de modelado. Madrid: Addison Wesley, 1999.

CHAVEZ, M. Sistema de análisis de registros de servidores proxy para usuarios, La Habana: Universidad de las Ciencias Informáticas, 2011.

CHUNG, L., et al. Non-functional requirements in software engineering. Springer Science & Business Media, 2012, p. 1.

CLARO, Magdalena. Impacto de las TIC en los aprendizajes de los estudiantes: estado del arte. 2010.

COMPUTER HOPE Free computer help since 1998 [en línea]. [Fecha de consulta: 12 diciembre 2017]. Disponible en: https://www.computerhope.com/jargon/l/log.htm

CROWDSOURCEDTESTING smarter software testing. Tipos de pruebas. Pruebas funcionales. ¿Qué son pruebas funcionales? [en línea]. [Fecha de consulta: 10 mayo 2018]. Disponible en: https://crowdsourcedtesting.com/es/pruebas-funcionales, 2018.

DATOS.gob.es reutiliza la información pública [en línea]. [Fecha de consulta: 6 diciembre 2017]. Disponible en: http://datos.gob.es/sites/default/files/doc/file/informe_herramientas_visualizacion.pdf

DEVELOPERLOVE.com [en línea]. [Fecha de consulta: 15 diciembre 2017]. Disponible en: http://developerlover.com/monitorizacion-logs-stack-elk-elasticsearch-logstash-kibana/

DÜRSTELER, J. C. The history of visualization. Inf@ Vis, 2003.

FERNÁNDEZ, F. C. Glosario básico inglés-español para usuarios de Internet. 1994. p.49.

FIELDING, R.; et al. "Hypertext Transfer Protocol --HTTP/1.1", RFC 2616, Junio 1999.

FLANAGAN, David. JavaScript: the definitive guide. “O’Reilly Media, Inc.", 2006.

GAMMA, E. Design Patterns: Elements of Reusable Object-Oriented Software. s.l.: Pearson Education, 2013.

GARCÍA, J. Un estudio comparativo de dos herramientas MDA: OptimalJ y ArcStyler. s.l.: Departamento de informática y sistemas, Universidad de Murcia, 2011.

GARCÍA, M. J. Regulación y autorregulación en Internet: el control de los contenidos y los datos en la LSSI. 12, [sede de la Agencia Catalana de Protección de Datos], noviembre 2005.

GAUCHAT, Juan Diego. El gran libro de HTML5, CSS3 y Javascript. Marcombo, 2012.

GÓMEZ-DOMÍNGUEZ, David; RUIZ-RODRÍGUEZ, Antonio Ángel y PEIS-REDONDO, Eduardo. La gestión de documentos electrónicos: requerimientos funcionales. El profesional de la información, 2003, vol. 12, no 2, p. 89.

HENAO, Cristian. CoDejaVu: ¿Qué son las Convenciones de Código? In: CoDejaVu [en línea]. [Fecha de consulta 5 abril 2018]. Disponible en: http://codejavu.blogspot.com/2014/04/que-son-las-convenciones-de-codigo.html, 12 abril 2014.

ITBLOGSOGETI. PRUEBAS DE CARGA Y ESTRÉS. [en línea]. [Fecha de consulta: 10 mayo 2018]. Disponible en: https://itblogsogeti.com/2017/03/23/pruebas-de-carga-y-estres/, 23 de marzo de 2017.

JIMÉNEZ, Antonio García y MESEGUER, Alfonso Palazón. VISUALIZACIÓN EN LA INFORMACIÓN. Aproximaciones al periodismo digital. Librería-Editorial Dykinson. 2007. vol. 13, p. 139.

KEVELL [en línea]. [Fecha de consulta: 13 diciembre 2017]. Disponible en: http://kevellspanish.readthedocs.org/es/latest/sawstats.html

LABRADA Oduardo, Evelyn y ARAGÓN Barreda, Yaniel Lázaro. Desarrollo del Módulo de Gestión de Reportes Estadísticos para el sistema AiresProxyAudit. Tesis (Ingeniero en Ciencias Informáticas). La Habana: Universidad de las Ciencias Informáticas, Facultad 1, 2013. p. 42.

LARMAN, Craig. UML y patrones: una introducción al análisis y diseño orientado a objetos y al proceso unificado. s.l.: Pearson Educación, 2003.

LIE, Hakon Wium y BOS, Bert. Cascading style sheets: Designing for the web. Addison-Wesley Professional, 2005.

MANUAL de Usuario Smart Keeper. 2da ed. La Habana: Universidad de las Ciencias Informáticas, Facultad 1, marzo de 2014.

MARTÍN ÁLVAREZ, L.O. y GARCÍA MARTÍNEZ, Y. Sistema de reportes de la navegación por Internet. Pregrado (Ingeniería en Ciencias Informáticas), La Habana: Universidad de las Ciencias Informáticas, 2007.

MOGOLLÓN, Reyes, et al. Diseño y Desarrollo de la Solución de Software de Gestión de Espacios Físicos en la Universidad Distrital. 2017. p. 22.

NETBEANS.2017. NetBeans IDE - The Smarter and Faster Way to Code [en línea]. [Fecha de consulta: 15 diciembre 2017]. Disponible en: https://netbeans.org/features/

PALENZUELA, O.B. ISAWEB. Monitoreo de Tráfico en Internet, Ciudad Habana, 2006.

POTENCIER, Fabien y ZANINOTTO, François. symfony. Potencier. org [en línea]. [Fecha de consulta: 16 diciembre 2017]. Disponible en: http://fabien. potencier. org/article/65/why-symfony, 2014.

PRESSMAN, Roger. Software Engineering: A Practitioner´s Approach, 2011.

SAAVEDRA LÓPEZ, Dismey, et al. Aplicación web para la realización de estudios farmacocinéticos, versión 2.0. Revista Cubana de Informática Médica, 2013, vol. 5, no 2, p. 118-131.

SÁNCHEZ, Ricardo y ECHEVERRY, Jairo. Validación de Escalas de Medición. en Salud. Rev. Salud Pública, 2004, vol. 6, no 3, p. 302-318.

SÁNCHEZ, Tamara Rodríguez. Metodología de desarrollo para la Actividad productiva de la UCI, La Habana: Universidad de las Ciencias Informáticas. s.f.

SHANNON, Ross. What is HTML. YourHTMLSource, c2000-2009 [en línea]. [Fecha de consulta: 15 diciembre 2017]. Disponible en: http://www. yourhtmlsource. com/starthere/whatishtml. html, 2007.

SINOLOGIC La web más leída sobre VoIP en Español [en línea]. [Fecha de consulta: 12 diciembre 2017]. Disponible en: https://www.sinologic.net/2014-02/como-sacar-una-traza-y-por-que-es-tan-importante.html

SOMMERVILLE, Ian. Ingeniería del software. Técnicas cuantitativas para la gestión en la ingeniería del software Pearson Educación, 2005, p. 55 y 56.

SOMMERVILLE, Ian. Ingeniería del software. 7ª ed. Madrid: Pearson Educación S.A, 2011. ISBN: 84-7829-074-5.

SQUID: Optimising Web Delivery. In: [en línea]. [Fecha de consulta 15 diciembre 2017]. Disponible en: http://www.squid-cache.org/

TECHTARGET SearchBusinessAnalytics [en línea]. [Fecha de consulta: 6 diciembre 2017]. Disponible en: http://searchbusinessanalytics.techtarget.com/definition/data-visualization

THE WEBALIZER [en línea]. [Fecha de consulta: 13 diciembre 2017]. Disponible en: http://www.webalizer.org/

UCI. [en línea]. [Fecha de consulta: 15 noviembre 2017]. Disponible en: http://www.uci.cu.

UNAD. Lenguaje Unificado de Modelado UML. Diagrama de Clases de Diseño [en línea]. Universidad Nacional Abierta y a Distancia, (2016). [Fecha de consulta: 9 de mayo de 2018]. Disponible en: http://stadium.unad.edu.co/ovas/10596_9836/diagrama_de_clases_de_diseo.html.

UNIFIED Modelling Lenguaje. [en línea]. [Fecha de consulta: 5 marzo 2018]. Disponible en: http://www.uml.org.

VALLES Cepeda, Monserrat. Validación de un Sistema de Información [en línea]. [Fecha de consulta: 10 mayo 2018]. Disponible en: https://prezi.com/gxj16_2khrwi/validacion-de-un-sistema-de-informacion/, 19 de mayo de 2014.

VISCONTI, Marcello y ASTUDILLO, Hernán. Fundamentos de Ingeniería de Software. Universidad Técnica Federico Santa María, 2011.