Ciencias Matemáticas, Vol. 35, No. 2, 2021, Pag. 81-90
Recibido 01-2021, Aceptado 09-2021

Desarrollo de la nueva versión multiplataforma de OpenLatino Server (OLS)
Develop of the new multiplatform version of OpenLatino Server (OLS)

Frankie Mujica Cal[1], Joanna Campbell Amos[1], Eduardo Quesada Orozco[1]

Resumen OpenLatino es un servidor de mapas desarrollado por el departamento Casasoft de la Universidad de la Habana. Este software fue desarrollado hace algunos años y actualmente presenta varias deficiencias que limitan su uso. En el siguiente trabajo se propone desarrollar una versión multiplataforma de *OpenLatino Server*, que posea, al menos, todas las funciones básicas que ya este tenía y que mejore sus deficiencias respecto a otros sistemas similares.

Abstract OpenLatino is a map server developed by the Casasoft department of the University of Havana. This software was developed a few years ago and currently has several deficiencies that limit its use. In the following work it is proposed to develop a multiplatform version of *OpenLatino Server*, which has at least all the basic functions that it already had and which improves its deficiencies compared to other similar systems.

Palabras Clave
SIG, ASP.NET Core, WMS, Multiplataforma, Framework

Keywords
GIS, ASP.NET Core, WMS, Multiplatform, Framework

[1] *Departamento de Programación e Ingeniaría de software, Universidad de la Habana, La Habana, Cuba, frankie.mujica@matcom.uh.cu, joanna@matcom.uh.cu, quesada@matcom.uh.cu*
***Autor para Correspondencia, Corresponding Author**

Introducción

Los Sistemas de Información Geográfica (**SIG** o **GIS**, por sus siglas en inglés, Geografic Information System) son un conjunto de herramientas informáticas que asocian elementos espaciales con bases de datos y permiten al usuario crear consultas interactivas, analizar mejor la información, editar datos, mapas y presentar los resultados de todas estas operaciones. Gracias al surgimiento y desarrollo de la internet se hizo más fácil compartir y actualizar la información, lo que motivó el surgimiento de los **Web GIS**[1]

Un componente esencial para un **Web GIS** es el Servidor de Mapas (en inglés conocido como **IMS**: *Internet Map Server*). Este es de gran importancia debido a que es capaz de manejar mucha información proveniente de diversos proveedores y permite realizar diversas consultas u operaciones (*Queries* en inglés) sobre los datos que maneja en solo unos milisegundos, incluso si se trata de millones de datos. Los servidores de mapas son los encargados de servir los datos alfanuméricos de los territorios y los datos relacionados con las geometrías de los lugares (también conocidos como Datos Espaciales). Para manejar los datos, estos son agrupados en conjuntos denominados Capas, de modo que un mapa es el resultado de combinar varias capas. Las capas a su vez pueden contener datos de dos tipos: Raster o Vectorial, los datos raster son, en esencia, cualquier tipo de imagen digital representada en mallas y los vectoriales están destinados a almacenar con gran exactitud los elementos geográficos, cada una de estas geometrías está vinculada a una fila en una base de datos que describe sus atributos y polígonos.

Como consecuencia de su importancia, el servidor de mapas constituye el objeto de estudio de este trabajo

1. Motivación , justificación y formulación del problema

OLS fue desarrollado usando las buenas prácticas de la programación orientada a objetos [1] y los principios **SOLID** [2] por lo que su código es fácil de entender y permite incorporar mejoras fácilmente. Por otro lado, fue creado usando la tecnología de *ASP.NET Framework*[2][4], en consecuencia, solo

[1] **Web GIS** es un concepto que engloba las funcionalidades de un sistema de información geográfico y permite acceder desde internet.

[2] Un *Framework* es un conjunto de bibliotecas de clases que engloban conceptos, prácticas y criterios para enfocar un tipo de problemática particular.

puede ejecutarse en entorno Windows y publicarse usando **IIS**[3] [5].

Esto lo pone en desventaja respecto a otros servidores multiplataforma (es decir, que pueden ejecutarse en varias plataformas como Windows y Linux) que cada vez dominan más el mercado del software debido a las ventajas de portabilidad[4] que ofrecen. Además, podemos señalar otros aspectos importantes que demuestran la necesidad de desarrollar una nueva versión de OpenLatino Server como:

1. Para funcionar bien, necesita una serie de configuraciones para sus capas y los proveedores de los datos de estas, así como los estilos que se pueden usar, su información alfanumérica, entre otras. Esas configuraciones se obtienen de la base de datos, pero no existe una herramienta visual que facilite el trabajo, por lo que el servidor solo puede ser configurado por un programador.
2. Para comunicarse con sus clientes, **OLS** utiliza el protocolo **WMS** [7][8], pero tiene pendiente la implementación del pedido *GetCapabilities* que envía información descriptiva de las capas que puede servir, los servicios con los que cuenta y los estilos con los que puede renderizar[5] las capas.
3. No existe una interfaz visual para registrar aplicaciones y administrar sus espacios de trabajo (en inglés *Workspaces*) y permisos sobre los recursos y funciones de **OLS**.
4. Antes de responder a las *Queries* hechas por una aplicación cliente, **OLS** usa otro servidor auxiliar de autorización para saber si el pedido debe ser respondido o denegado su acceso. Sin embargo, el servicio de autorización pudiera realizarse desde el mismo *OpenLatino* sin necesidad de tener otro servidor ejecutándose paralelamente que es más costoso económicamente y computacionalmente.
5. El pedido *GetMap* del protocolo **WMS** funciona correctamente, pero en ocasiones, puede ser muy lento, por lo que necesita alguna mejora de rendimiento.

Podemos decir que es posible el desarrollo de una nueva versión multiplataforma de **OLS** que mejore los aspectos señalados, debido a que, en la actualidad, ya existen otros servidores multiplataforma como: *QGis Server*, un servidor multiplataforma desarrollado en *Python* y publicado en Apache [6] que es muy popular en el contexto internacional, entre otras cosas: por ser eficiente, configurable y su alta portabilidad. Además, es posible migrar códigos de *ASP.NET Framework* a *ASP.NET Core*[6], una tecnología más reciente de Microsoft que permite desarrollar servidores Web multiplataforma, y así aprovechar algunas de las funcionalidades ya existentes en **OLS** para no tener que implementarlas desde cero y así ahorrar tiempo y esfuerzo.

Dadas las deficiencias que posee el servidor actual de OpenLatino, podemos definir el problema a partir de las siguientes interrogantes:

1. ¿Será posible, usando *ASP.NET Core*, desarrollar un nuevo servidor capaz de funcionar de forma estable tanto en Windows como Linux?
2. ¿Será posible migrar los códigos para *ASP.NET Core* de, al menos el núcleo de **OLS**, que se encarga de obtener información de fuentes de datos vectoriales y es capaz de convertirla, a través de un proceso de *renderizado*, en imágenes en formato PNG o JPG para responder los pedidos *GetMap*?
3. ¿Cómo hacer un módulo con interfaz visual cómoda para configurar las capas, los estilos, los proveedores y demás configuraciones que requiere el servidor?
4. ¿Cómo hacer un módulo con interfaz visual para registrar aplicaciones y gestionar sus *Workspaces*?
5. ¿Cómo implementar la función *GetCapabilities* del protocolo **WMS** respetando la arquitectura que ya existía y manteniendo buenas prácticas de programación?
6. ¿Cómo integrar un módulo de seguridad a la interfaz de **OLS** que, de forma eficiente, sea capaz de determinar cuándo un pedido es atendido y cuando es ignorado?
7. ¿Qué mejoras pueden hacerse al pedido GetMap para que responda de modo más rápido?

1.1 Objetivos

Para resolver los problemas presentados y responder a las interrogantes, se plantean los siguientes objetivos principales:

1. Implementar las funciones básicas del nuevo servidor basándose en el aprovechamiento de las buenas prácticas y la correcta arquitectura que ya posee *OpenLatino Server*.
2. Implementar la función *GetCapabilities* y completar así todas las funciones del protocolo **WMS**, basándose en el estándar establecido por el Open Geospatial Consortium (**OGC**) [9].
3. Implementar un módulo de interfaz visual que permita, a los administradores de **OLS**: agregar, eliminar o editar las capas del servidor, así como los estilos, los proveedores de datos y la información alfanumérica. Además de permitir, a usuarios comunes, registrar sus aplicaciones y crear, editar o eliminar *Workspaces* para las mismas.
4. Implementar un módulo de seguridad que se acople al servidor y permita determinar de forma eficiente cuando responder o ignorar los pedidos provenientes de una aplicación cliente.

[3] *Internet Information Service* (**IIS**) es un Servidor Web y un conjunto de servicios para el sistemea operativo Windows.

[4] Se entiende por Portabilidad a la capacidad de un software de ejecutarse en diferentes plataformas

[5] Renderizar es un término que no posee traducción al español, pero se refiere a generar imágenes a partir de objetos en memoria.

[6] Es un servidor HTTP de código abierto que funciona en Linux, Windows o Mac.

5. Mejorar la eficiencia del proceso de renderizado del servidor para que la respuesta al pedido *GetMap* sea más rápida.

Para cumplir con los objetivos principales explicados previamente, se plantean los siguientes objetivos secundarios:

1. Investigar las ventajas y analizar la factibilidad de migrar los códigos ya existentes y la arquitectura de **OLS** a la nueva tecnología *ASP.NET Core*.
2. Investigar las especificaciones del protocolo **WMS** que se establece en el **OGC** para una correcta implementación del protocolo *GetCapabilities*.
3. Analizar las variantes existentes para desarrollar la interfaz de usuario de la configuración del servidor y registro de aplicaciones.
4. Definir mejoras a la arquitectura de **OLS** que permitan incorporar fácilmente el nuevo módulo de seguridad.

2. Estado del Arte

En la década de los 90 se hicieron populares los Sistemas de Información Geografica gracias a que ya las computadoras eran más potentes y podían procesar mayor cantidad de datos y también gracias al surgimiento de *ArcView*, un software de escritorio que era capaz de usarse en Windows y su difusión propició la adopción de su estándar de archivos (*Shapefile*) como estándar internacional para los **SIG**. Además, el surgimiento de la internet propició un medio para extender los mapas y dio lugar al surgimiento de los servidores de mapas.

En la actualidad es prácticamente imposible que un **SIG** tenga éxito sin el uso de servidores de mapas web debido a las importantes ventajas sobre las versiones de escritorio que ofrecen, por ejemplo: estos brindan la posibilidad de obtener información más actualizada, incluso en tiempo real y que los usuarios no necesitan instalar ningún software especializado para obtener y visualizar los datos.

2.1 Frameworks y Librerías de mapas

Para crear la nueva versión de **OLS** con las funcionalidades antes mencionadas, se analizaron distintas bibliotecas y herramientas que pudieran servir para implementarlas. A continuación, se describen algunas de las variantes.

***SharpMap*[11]** : Es una biblioteca de mapas para crear aplicaciones web y de escritorio, de la cual se posee conocimiento previo en *Casasoft*. Con esta librería se pueden realizar consultas a los datos espaciales para el manejo y análisis de los mismos.

Ventajas: Genera mapas de alta calidad para ser visualizados en la aplicación cliente. Soporta una amplia variedad de formatos de entrada como: *PostGIS*, *Shapefile*, *ArcSDE*, *OracleSpatial*, *SpatiaLite*, *MapInfo File*, entre otros. Es extensible a diversos tipos de proveedores de datos.

Desventajas: Los mecanismos de comunicación con el servidor son muy cerrados, por lo que hacer cualquier modificación en el funcionamiento de un protocolo representa un problema y hace el código poco extensible.

Conclusión:En general, tiene todo lo básico que se necesita, pero fue desarrollado usando *C#* basado en el *Framework .NET 4.0* y no existe una versión que sea compatible con *ASP.NET Core*, por lo que NO es multiplataforma y hay que descartarlo.

***ArcGis* [12] [13]** : Es un conjunto de productos de software producidos y distribuidos por ESRI[10]. La familia de aplicaciones de escritorio, es una de las más ampliamente utilizadas, incluyendo en sus últimas ediciones las herramientas *ArcReader*, *ArcMap*, *ArcCatalog*, *ArcToolbox*, *ArcScene* y *ArcGlobe*, además de diversas extensiones. Recientemente fue anunciada una nueva versión que incluye componentes visuales para leer y mostrar mapas para **WPF** en *ASP.NET Core 3.0*.

Ventajas: Se ajusta a los estándares internacionales establecidos en la OGC lo que garantiza que un mayor número de clientes puede acceder a él (Pues la mayoría de los clientes usa el estándar de la **OGC**).

Desventajas: El software no es gratuito, sino que se distribuye comercialmente bajo tres niveles de licencias que son, en orden creciente de funcionalidades (y coste): *ArcView*, *ArcEditor* y *ArcInfo*.

Conclusión: Esta opción se descarta porque la componente que tiene disponible multiplataforma sólo funciona en entorno de escritorio (**WPF**) y el objetivo es hacer un servidor Web. Además, no es gratuito.

***AsposeGIS* [14]** : Este *Framework* permite acceder y manipular información geográfica procedente de fuentes vectoriales. Su interfaz de aplicaciones (**API** por sus siglas en inglés) se encuentra disponible para *ASP.NET Core*.

Ventajas: Como se encuentra disponible en *ASP.NET Core* permite crear un servidor Web multiplataforma como se requiere. Permite leer, escribir y convertir varios de los formatos más comunes en los **GIS** como: *Shapefile*, *GeoJSON*, *FileGDB*, *KML*, entre otros. Además, permite renderizar a formatos de imagen como: JPEG, PNG, BMP, entre otros y permite realizar consultas espaciales tales como: calcular la intercepción de objetos, obtener el centroide de una geometría, encontrar la distancia mínima entre dos geometrías, componer dos polígonos, entre otras.

Desventajas: No posee soporte para el protocolo **WMS**. No es gratuito.

Conclusión: Esta opción no es viable, pues no es gratuita. Además, no implementa el protocolo **WMS** que se requiere y sí está implementado en **OLS**.

***Map Tile Service* [15]** : Es una implementación minimalista y a código abierto de un servidor de mapas capaz de servir capas *Raster*[7] provenientes de bases de datos o archivos locales. Fue desarrollado en 2017, en C# y en *ASP.NET Core* 2.0.

Ventajas: Esta opción podría proporcionar un punto de partida para desarrollar la nueva versión que queremos, es multiplataforma y brinda lo necesario para el manejo de capas *Raster*.

Desventajas: Al ser una implementación tan básica y sencilla, carece de muchas de las características que ya posee **OLS**. Ejemplo: No implementa el protocolo **WMS**, no permite leer de fuentes de datos vectoriales, no tiene implementadas operaciones espaciales.

Conclusión: Se descarta esta opción debido a que, por carecer de tantas características (ya existentes en **OLS**), habría que reimplementar muchas funciones, algunas bien complejas, como renderizar capas provenientes de fuentes de datos vectoriales a formato PNG o JPG y ocuparía demasiado tiempo.

Debido a que no se encontró una variante de *.NET* que sea factible para cumplir los objetivos planteados. Se analizó la posibilidad de utilizar otros Frameworks Web para implementar las funciones básicas del **IMS** reutilizando la mayor cantidad de código posible de OLS. De esta manera, se pueden cumplir los objetivos planteados manteniendo (en su mayoría) el modelo y las buenas prácticas establecidas en **OLS** y no se desecha el trabajo ya existente. Entre los Frameworks estudiados se encuentran: *Django*, *Laravel*, *Express* y *ASP.NET Core*, pero se decide utilizar *ASP.NET Core* debido a su similitud con *.NET Framework*, que es con el que se desarrolló OLS. Esto permite reutilizar gran parte del código original. Además, *.NET Core* es un Framework de gran eficiencia, multiplataforma, gratuito y de código abierto. Se pueden encontrar los detalles de esta elección en [26].

3. Proceso de desarrollo

El primer paso para obtener las nueva versión de OLS fue reutilizar la mayor parte posible de la versión existente. Para ello se llevó a cabo un proceso de migración del código, proceso engorroso que lleva analizar todas las dependencias de cada módulo y realizar una serie de pasos para adaptar el código para el nuevo Framework, los detalles de este proceso se pueden ver en [26]. Una vez realizado el proceso, se tiene cumplido el primer objetivo, pues, se cuenta con una nueva versión en *.NET Core* que posee las funciones básicas **OLS**. Además, se mantiene la misma arquitectura del servidor y tiene casi el mismo modelo de datos (las diferencias son los pequeños cambios que se hicieron para adaptarse a *EF Core*).

Para agregar las nuevas funcionalidades fue necesario definir un nuevo modelo de datos basado en el anterior pero con modificaciones para la seguridad y para acoger el nuevo concepto de Espacio de Trabajo. Se separó el concepto de Usuario del de Cliente. Un Usuario representa al humano que accede a la interfaz visual y puede hacer determinados cambios como agregar, editar o eliminar capas. Un Cliente representa una aplicación registrada para usar los servicios del servidor de mapas. A nivel de base de datos se representan con las tablas *ApplicationUser* y *Client* respectivamente. Otro cambio fue en el tipo de relación entre Capas y Estilos. Anteriormente, una Capa podía tener múltiples estilos, pero un estilo solo podía estar asociado a una Capa. Esto tiene como consecuencia negativa, que obliga a crear al menos un estilo por cada capa, siendo, a veces, visualmente idénticos. En cambio, se decide que esta relación debe ser de muchos a muchos, lo que permite que un mismo estilo sí puede tener varias capas que lo usan.

3.1 Nuevo concepto de *Workspace*

En el servidor de seguridad que usa **OLS**, existe un concepto de *Workspace*. Este define, para cada usuario, todos los permisos que posee. Los permisos están definidos por las entidades *ServiceAccessRule* y *DataAccessRule* que a su vez guardan información de una entidad *Permisssion* que puede, a su vez, tener permisos heredados. Por lo que: autorizar un determinado Usuario, con un determinado *Workspace*, para que pueda hacer un *Request* a **OLS** se puede volver complicado.

En esta nueva versión, se redefinió la entidad *Workspace*. En esencia mantiene su objetivo, establecer los permisos que tienen los clientes, pero también agrega personalizaciones de estilo a las capas para cada cliente. Ahora un *Workspace* tiene asociada una lista de capas y una lista de funciones(*IFunction*), además existe una relación de muchos a muchos entre Cliente y *Workspace* que para representarla requiere otra tabla. La nueva tabla que se requiere para manejar la relación muchos a muchos, también tiene asociado un Estilo que permite personalizar como debe pintarse cada Capa para ese Cliente. En la figura 1 se muestra la estructura final con que quedó la nueva versión de **OLS**, las nuevas tablas que se agregaron están desplegadas.

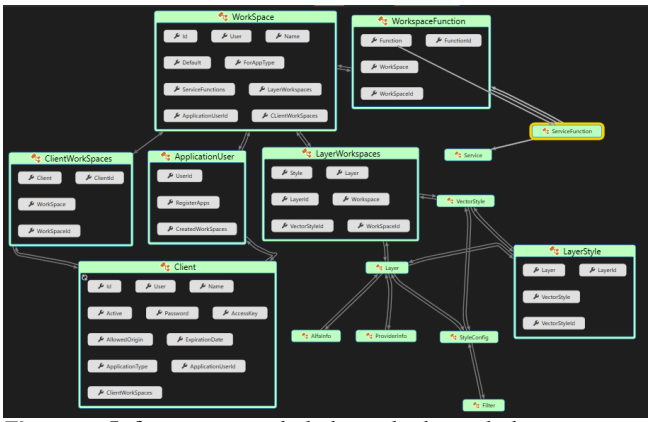

Figura 1. Infraestructura de la base de datos de la nueva versión de **OLS**.

[7]Una capa *Raster* es una imágen en mapa de bits, con una estructura en forma de rejilla rectangular de pixeles.

Los datos a los que se tiene acceso pueden definirse por las capas a las que se tiene acceso, en otras palabras, un cliente solo podrá acceder a los datos contenidos en las capas a las que puede acceder.

Los servicios a los que se tiene acceso están definidos por los *IFunction* que se pueden solicitar. Los servicios, son los tipos de pedidos que se pueden hacer, por ejemplo: *GetMap*, *GetCapabilities*, entre otros, y por cada tipo de pedido existe un *IFunction* que lo representa. Es importante notar, que los permisos de lectura o escritura también están definidos por los *IFunction*, ya que una función tiene implícitas operaciones de lectura o escritura, por ejemplo: *GetMap* solo hace operaciones de lectura. Si se quiere dotar a un cliente de permiso "total"para lectura, bastaría con permitirle acceder a todos los *IFunction* que realizan operaciones de lectura.

Todo cliente que se registra en el nuevo servidor de *OpenLatino* adquiere acceso a un *Workspace* llamado çommon". Este espacio de trabajo cuenta con un mínimo de funcionalidades y capas para que el cliente pueda usar. Además, el usuario que registró la aplicación puede crear nuevos *Workspaces* y así darle acceso a más capas y servicios.

Para comunicarse con el servidor, el cliente debe incluir, en los *Headers* de sus pedidos, el Identificador del *Workspace* por el que quiere que se responda su pedido. Se pueden ver los detalles en [27]

El uso del nuevo concepto de *Workspace* trae consigo cambios importantes en las funcionalidades del servidor, por ejemplo: *GetCapabilities* devuelve un fichero XML[8] con la información referente a todos los servicios que se pueden solicitar y las capas y estilos disponibles. Pero, ahora estas están restringidas a solo lo que es accesible por el espacio de trabajo que se especifica en el pedido (Los detalles del proceso de implementación pueden verse en el capitulo 4 de [26]). El pedido *GetMap* tiene una diferencia importante en cuanto al estilo con el que se dibujan las capas respecto a como se hacía en **OLS**. Ahora el estilo de una capa se determina de la siguiente forma:

1. Se pinta con el estilo solicitado en el pedido, si la propiedad *Styles* del mismo incluye uno por cada capa solicitada y cada capa posee el estilo que se le solicita.
2. Se pintan todas las capas con el estilo que tienen definido por defecto en el *Workspace* solicitado, si no se puede aplicar (1).
3. Para las capas que no se aplique (1) y no tienen un estilo por defecto en el *Workspace* solicitado, se escoge el primer elemento en la lista de estilos de esa capa.

3.2 Pedido GetCapabilities

Otro de los objetivos planteados es completar la implementación del protocolo WMS segun se especifica en su sitio oficial de **OGC**: https://www.opengeospatial.org/standards/wms. Para ello, es necesario implementar el pedido *GetCapabilities*. El propósito de este pedido es obtener metadatos de los servicios que ofrece el servidor. Los metadatos son especificados en formato XML que es un formato entendible tanto por humanos como por aplicaciones.

La información requerida y el formato en que debe aparecer, se encuentran bien definida en la especificación de **WMS** que se mencionó previamente. Algunos de los datos más importantes que contiene son:

- La versión del protocolo que se está usando.
- Nombre del servidor de mapas y una pequeña descripción del mismo.
- El número máximo de capas que se pueden solicitar en un mismo pedido.
- El tamaño máximo que puede solicitarse para una imagen de mapa.
- La lista de pedidos del protocolo **WMS** que están implementados en el servidor.
- La lista de todas las capas que se pueden solicitar con sus posibles estilos.

Los detalles sobre su implementación pueden encontrarse en el capitulo 4 de [26].

3.3 Seguridad a nivel de Cliente

Para la seguridad a nivel de Clientes, es necesario establecer un protocolo de comunicación que permita identificar, desde el propio pedido, qué aplicación es la que hace el pedido y qué permisos tiene. Para lograr esto, se decide implementar el protocolo **JWT**.

Jason Web Token (**JWT**) se especifica en el estándar RFC 7519. Este define una forma compacta y auto-contenida de asegurar la transmisión de información entre dos partes usando el formato JSON. Esta información es confiable porque se encuentra firmada de forma digital. La firma digital puede hacerse usando una llave secreta (con el algoritmo HMAC[9]) o con un par de llaves pública/privada usando RSA[10] o ECDASA[11]. Además, los **JWT** pueden ser encriptados, para que la información contenida en él sea ilegible para cualquiera que intercepte el pedido, cuando el token fue firmado usando un par de llaves publica/privada, esto también asegura que el Token solo puede ser decodificado por quien lo generó.

Cuando se registra una nueva aplicación cliente, basado en la información de esta, se genera un token único que se le muestra al Usuario que registró la aplicación. Ese Token debe ser enviado en los *headers* de cada pedido que la aplicación haga a **OLS**. Cada vez que llegue un pedido al nuevo servidor de OLS, se valida que este contenga un token válido y que el cliente que hizo la solicitud tiene los permisos necesarios.

[8] XML es un formato estándar para el intercambio de información estructurada entre diferentes plataformas.

[9] HMAC es un algoritmo usado en Criptografía para calcular el código de autentificación de un mensaje

[10] Sistema Criptográfico de clave pública.

[11] Algorítmo Criptográfico propuesto como mejora del algoritmo **DSA**

Problema

Aun con todas las comprobaciones que se hacen, hay un problema grave de seguridad, pues, un agente externo puede interceptar los pedidos que viajan a **OLS** y obtener un token que no es suyo, esto implica que puede usarlo para realizar pedidos al servidor como si fuera ese cliente y acceder a información confidencial sin ser detectado.

Solución

Se le establece un tiempo de vida limitado para cada Token, es decir, cada uno de los Tokens que se les entrega a los clientes será válido solo por un tiempo prefijado. Este cambio garantiza que, si un Token es robado, este no servirá por mucho tiempo y una vez que se acabe su tiempo de vida no servirá. Para poder hacer este cambio, es necesario dotar a los Clientes auténticos de un método que les permita renovar sus Tokens para poder seguir accediendo a **OLS**, el proceso de actualización de tokens se explica detalladamente en [26].

3.4 Seguridad a nivel de Usuarios

ASP.NET Core provee varias funcionalidades como parte de su *Middleware*, una de ellas facilita integrar la Autenticación y Autorización de Usuarios al servidor. Con *Identity System* se hace muy fácil implementar la seguridad a nivel de Usuarios. Este sistema es fácilmente configurable, extensible y soporta varias formas de almacenar la información de cada usuario.

En este caso, se extiende la base de datos de la nueva versión de **OLS**, para que almacene allí también las tablas que requiere *Identity*.

3.5 Optimización del pedido GetMap

El pedido más lento, en **OLS**, es *GetMap*. Esto se debe a que requiere que, por cada una de las capas solicitadas, se obtengan sus geometrías, que usualmente son cientos o miles. Las geometrías se obtienen por medio de una consulta a base de datos y una vez conseguidas todas, se pasa a renderizar la imagen. El proceso de *renderizar* la imagen consiste en obtener una imagen, en formato PNG o JPG (usando *System.Drawing.Common*) de los polígonos obtenidos, componiendo y superponiendo cada uno de estas. En la antigua versión de **OLS**, esta solicitud demora como promedio alrededor de 4 segundos en dar una respuesta, lo que resulta demasiado lento.

Una mejora que se hacía era crear una *Cache* para guardar los bytes de cada imagen que se generaba. Luego de obtener las geometrías, se consultaba si la imagen de las mismas se encontraba guardada en *Cache*, en caso positivo se devolvía como resultado los datos de la *Cache*. Pero esta mejora realmente no aportaba mucho, porque solo agilizaba el proceso de *renderización*. Este proceso se *realiza* solamente con interacción de la memoria RAM y el Procesador que son capaces de procesar a altas velocidades incluso miles de Polígonos.

En cambio, la etapa de obtención de las geometrías es mucho más lento, pues, en las consultas a base de datos interviene el Disco Duro que es mucho más lento que la RAM y para obtener los miles de Polígonos que se solicitan usualmente, puede tardar unos segundos.

En la nueva versión de *OpenLatino* se reestructuró la Cache para que no solo optimice el proceso de renderizado, sino también la obtención de geometrías. Es evidente que para pedidos donde se solicitan las mismas capas, se obtienen los mismos Polígonos y por eso, con la nueva *Cache*, primero se pregunta si está guardada la imagen para las capas y estilos que se solicitan y, en caso positivo, se da como respuesta la imagen guardada.

De esta forma se evita tener que obtener las geometrías de forma innecesaria lo que provoca que el tiempo de respuesta para capas y estilos que ya hayan sido solicitados disminuya a solo 0.1 segundos.

Para implementar la nueva *Cache* se usó MongoDB, que es el mismo gestor de base de datos (No Relacional) que se usaba en **OLS**. Esto se debe a que se siguen almacenando los Bytes de las imágenes que se generan y no es posible usar una base de datos relacional para almacenar cadenas de Bytes de gran tamaño [17]. Los detalles sobre la estructura de la nueva *Cache* pueden encontrarse al final del capítulo 3 en [26].

3.6 Interfaz Visual de Administración

ASP.NET Core posee una implementación del patrón **MVC** [6] (Modelo-Vista-Controlador o *Model-View-Controller*, en inglés), esto agiliza mucho el proceso de crear interfaces visuales para las aplicaciones Web y por esto, unido a la experiencia previa con esta tecnología, se decide crear la interfaz visual del módulo de administración de la nueva versión de **OLS** usando *ASP.NET Core MVC*.

Patrón MVC

A grandes rasgos, se puede decir que el patrón MVC significa separar el proyecto en, al menos tres partes:

- **Modelos**, que contiene o representa los datos con los que el usuario trabaja.
- **Vistas**, que son usadas para renderizar algunas partes del modelo como una interfaz visual.
- **Controladores**, que procesan los pedidos que le llegan al servidor, realizan operaciones con el modelo y seleccionan qué vistas deben ser mostradas al usuario.

Cada parte de la arquitectura está bien definida y autocontenida, gracias, a la separación de conceptos. La lógica que manipula los datos del Modelo se encuentra solo en el Modelo, la lógica de mostrar contenido visual se encuentra solo en las vistas y el código que maneja los pedidos del usuario se encuentra solo en el controlador. Con esta clara separación de las partes, la aplicación, en este caso **OLS** será extensible y fácil de mantener sin importar que tan grande se haga.

4. Experimentación y Resultados

En el siguiente capítulo se muestran un conjunto de pruebas de funcionalidad y de rendimiento que se hicieron, en la

nueva versión del servidor de *OpenLatino Server* para probar que se cumplieron los objetivos planteados.

Para la realización de las pruebas se utilizó una laptop con las siguientes características:

- Procesador AMD A10-7300 Raedon R6, 1900Ghz
- 8Gb de RAM
- Disco duro Toshiba 1Tb, Serial ATA, 5400 Rpm

Para simular pedidos de una aplicación cliente al servidor, se usó Postman[19].

Pruebas de Funcionalidad

Prueba 1: Prueba de Registro de Aplicaciones y Workspaces

Objetivo

Mostrar el correcto funcionamiento del módulo visual para el registro de aplicaciones y configuración de *Workspaces*.

Para esta prueba se registró un Usuario común en el servidor (usuariotest@gmail.com), se creó un espacio de trabajo personalizado (TestW1) con acceso a los pedidos *GetMap* y *GetCapabilities* y a las capas Calles y Edificios solamente. Se registró una aplicación (AppTest1) y se hizo un pedido *GetMap* para solicitar las capas Calles y Edificios.

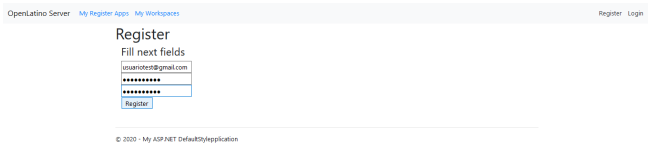

Figura 2. Registro del usuario (usuariotest@gmail.com).

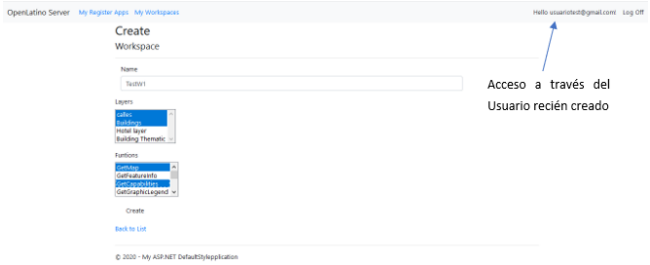

Figura 3. Creación del *workspace* TestW1.

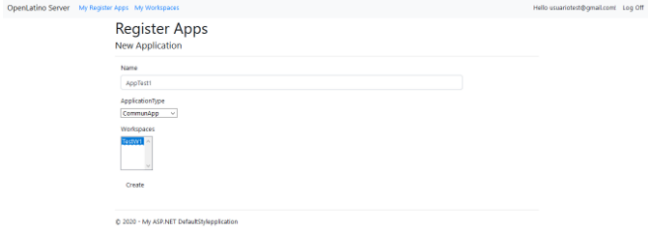

Figura 4. Registro de la aplicación AppTest1.

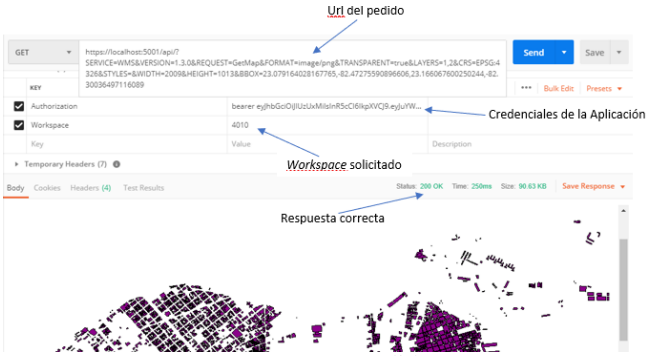

Figura 5. Respuesta a pedido *GetMap*.

Resultado

Con esta prueba se comprobó el correcto funcionamiento del Registro y Login de usuarios que forman parte de la seguridad a nivel de usuario, además, se comprobó el correcto funcionamiento de la interfaz visual para la creación de espacios de trabajo y registro de aplicaciones. Por último, la respuesta positiva del pedido GetMap comprueba que funcionan correctamente las funciones básicas del servidor.

Prueba 2: Funcionamiento en Linux

Objetivo

Comprobar que la nueva versión de **OLS** es capaz de funcionar también en el sistema operativo Linux.

Para esta prueba, se pondrá a ejecutarse el servidor en Ubuntu 18.04, en la misma laptop que se mencionó al principio del capítulo, se intentará acceder desde el navegador a la página de inicio del servidor y se hará un pedido *GetCapabilities*.

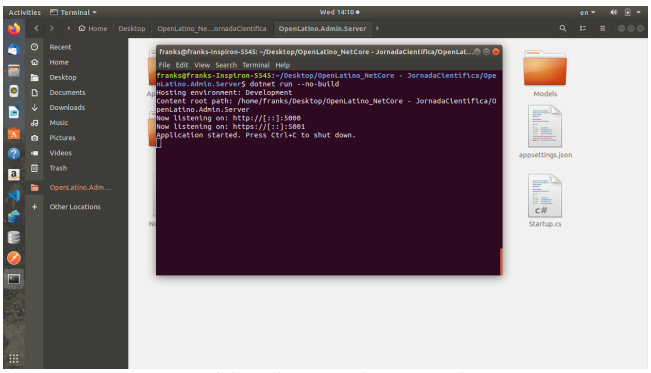

Figura 6. Servidor ejecutándose en Ubuntu 18.04.

Resultado

Con esta prueba se mostró que la nueva versión de OLS es capaz de funcionar correctamente tanto en el Sistema Operativo Windows como en Linux sin realizar ningún cambio significativo.

Pruebas de Rendimiento

Uno de los objetivos planteados en este trabajo fue mejorar la eficiencia del pedido *GetMap* que es el más lento en la

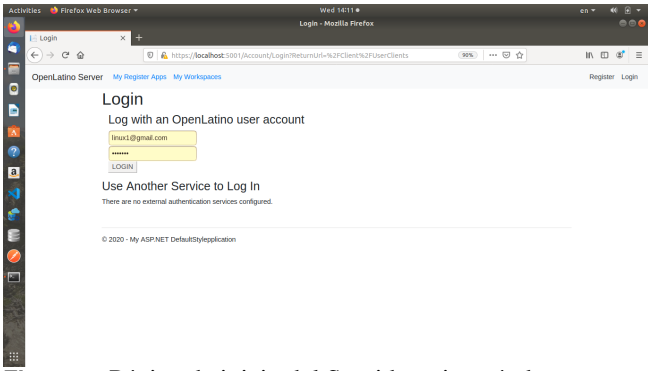

Figura 7. Página de inicio del Servidor, ejecutándose en Ubuntu 18.04.

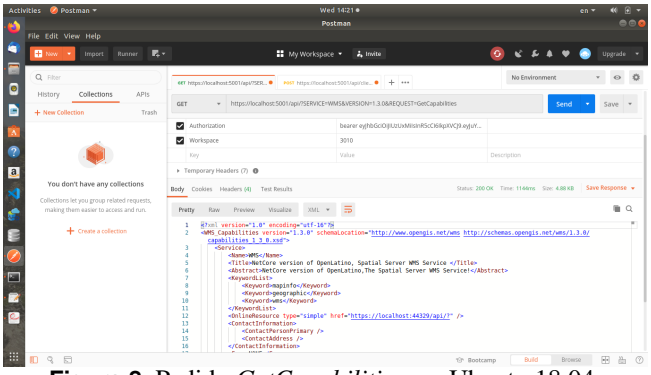

Figura 8. Pedido *GetCapabilities*, en Ubuntu 18.04.

antigua versión de **OLS**. A continuación, se exponen varias pruebas realizadas para medir la eficiencia del nuevo servidor comparándolo con la antigua versión o con otro servidor desarrollado en *Casasoft* que fue hecho en *.Net Framework* usando *SharpMap* para trabajar con mapas.

Para llevar a cabo las pruebas, se utilizó un software escrito en Go [20] llamado GoBench [21], el cual permite definir un *Request* y crear N procesos, cada uno simula ser un cliente que hace constantemente el pedido que se especificó por un tiempo T y luego reporta estadísticas útiles sobre el rendimiento del servidor al que se le realizaron las pruebas.

Prueba 1:

Objetivo

Saber si la nueva versión de **OLS** mejoró su velocidad de respuesta en pedidos básicos que no requieren interacción con base de datos. Para esta prueba, se hace un pedido para que muestre la página de inicio y se simulan 500 clientes.

	Antigua versión	Nueva versión
Num Pedidos Realizados	35594	102421
Respuestas a tiempo	35594	102421
Num Respuestas por Seg	1227	2926
Tiempo de Prueba (Seg)	30	30

Figura 9. Prueba de rendimiento #1.

Resultado

La disponibilidad del servidor se mantiene en ambos casos, pero la versión nueva supera con creces a la anterior respecto a la cantidad de pedidos que es capaz de responder y el tiempo que le toma en responder un pedido.

Prueba 2:

Objetivo

Saber si nueva versión de **OLS** mejoró la eficiencia del pintado del mapa. Para esta prueba, se hace un pedido *GetMap* que nunca antes se había hecho (No está la respuesta en la *Cache* inicialmente) sobre las capas 1, 2 y 3 a las dos versiones de **OLS** y se simulan 2 clientes por 30 segundos.

	Antigua versión	Nueva versión
Num Pedidos Realizados	8	115
Respuestas a tiempo	4	115
Tiempo Prom de Resp(Seg)	4.30	0.17
Num Resp por Seg)	0	5

Tabla 1. Prueba de rendimiento #2

Resultado

OLS mejoró su disponibilidad ya que antes no era capaz de responder a todos los pedidos que le hacían los clientes en un tiempo corto y ahora si mantiene esta disponibilidad. Además, el tiempo de respuesta disminuyó drásticamente, cuando se repite el pedido varias veces, lo que implica una mejora importante.

Con las pruebas de funcionalidad y rendimiento, se pudo comprobar que la nueva versión de **OLS** cumple con todos los objetivos propuestos. Además de las pruebas anteriormente descritas se hicieron otras, se pueden encontrar los detalles de todas las pruebas realizadas en el capítulo 5 de [26].

5. Conclusiones

Con el trabajo realizado, se han cumplido todos los objetivos planteados inicialmente. Pues, se obtuvo una nueva versión multiplataforma de **OLS**, con todas las funcionalidades básicas de la antigua versión. Además, se agregó un módulo visual para facilitar la configuración del servidor y uno para que los usuarios puedan registrar aplicaciones y configurar sus espacios de trabajo. Se mejoró la estructura que existía en el servidor para definir un espacio de trabajo, dando lugar a un nuevo concepto de *Workspace* que permite configurar el acceso a capas y funciones del servidor, así como personalizar estilos para las capas. Se implementó una capa de seguridad que maneja el acceso a los servicios de **OLS**, para que solo puedan acceder aplicaciones registradas y tiene en cuenta los permisos del Workspace que se solicita. Por último, se implementó una *Cache* que mejora notablemente el rendimiento del pedido *GetMap*.

Podemos concluir que los principales aportes de esta investigación son:

- El nuevo servidor de mapas desarrollado en *ASP.NET Core* es completamente funcional por sí mismo como servidor de mapas, pero, además, su capa de Dominio, por estar bien desacoplada, puede usarse como Framework para desarrollar nuevos servidores de mapas en *ASP.NET Core*. Usándose como *Framework*, brinda facilidades para procesar capas vectoriales y cumplir con el protocolo **WMS**. Facilidades que no estaban disponibles previamente para *ASP.NET Core* de forma gratuita.
- El estudio realizado, sobre el proceso de migración, representa para el futuro una fuente de referencia cada vez que se requiera un proceso similar.
- El proceso de implementación del pedido *GetCapabilities* constituye una guía para saber en el futuro como extender los protocolos de comunicación que implementa **OLS**.
- El módulo de seguridad implementado usando JSON Web Tokens provee a **OLS** de un mecanismo de seguridad robusto y extensible.
- El mecanismo de *Cache* implementado mejora mucho la eficiencia del pedido *GetMap* y sugiere otras nuevas mejoras que se pueden hacer en un futuro (Se explica mejor en el Trabajo Futuro).

5.1 Líneas futuras de investigación

A pesar de haber cumplido con los objetivos planteados inicialmente, el nuevo servidor de *OpenLatino* puede continuar mejorando, para ello, se proponen los siguientes aspectos como trabajo futuro:

- Implementar un proveedor de datos usando *PostgreSQL* [22] que permita procesar capas vectoriales y obtener las geometrías, ejecutándose desde Windows o desde Linux.
- Implementar proveedores de datos para formatos como *GeoJSON*, *ShapeFile*, *KML*, entre otros.
- Implementar un proceso que se ejecute en segundo plano y que genere respuestas a diferentes pedidos *GetMap* para que estas sean almacenadas en *Cache* e incremente las probabilidades de que cuando llegue un pedido *GetMap* al servidor la respuesta a este ya esté almacenada.
- Extender el uso de la *Cache* para que si se solicita una caja contenedora **A**, cuyas geometrías no están en *Cache*, pero pueden obtenerse por composición de las geometrías de las cajas contenedoras **B** y **C**, entonces, en lugar de consultar a base de datos para obtener todas las geometrías de **A**, se obtengan componiendo las geometrías de **B** y **C**.

Referencias

Referencias

[1] Meyer, Bertrand. *Object-Oriented Software Construction.* 1988.

[2] Eric Freeman y Elisabeth Freeman *Head First Design Patterns*. (First Edition), 2007. O'Really.

[3] Bipin Joshi *Beginning SOLID Principles and Design Patterns for ASP.NET Developers*. 2016 Apress

[4] Jamie Kurtz *ASP.NET MVC4 and the Web API.* , 2013. Apress.

[5] Sitio Oficial de Windows. URL: https://www.microsoft.com. Consultado en 14 de febrero de 2022

[6] Adam Freeman *Pro ASP.NET Core MVC*. (6th edition), 2016. Apress.

[7] Jeff de La Beaujardière *Web Map Service Implementation Specification* Part 2 XML for Requests using HTTP POST, 2002 OpenGIS Discussion Paper

[8] Jeff de La Beaujardière *OpenGIS® Implementation Specification* Version: 1.3.0, 2006 Open Geospatial Consortium Inc

[9] **OGC**. OGC official site. ESRI. URL: https://www.ogc.org/. Consultado en 14 de febrero de 2022

[10] **ESRI**. ESRI official site. ESRI. URL: https://www.esri.com/es-es/home. Consultado en 14 de febrero de 2022

[11] Codeplex. *SharpMap* - Geospatial Application Framework for the CLR. https://archive.codeplex.com/?p=sharpmap. Consultado en 14 de febrero de 2022

[12] **ESRI** official site. About *ARCGis*. https://www.esri.com/es-es/arcgis/about-arcgis/overview. Consultado en 14 de febrero de 2022

[13] **ESRI** Community *ARCGis*. https://comminity.esri.com/community/developers/native-app-developers/arcgis-runtime-sdk-for-net/blog/2019/08/22/announcing-arcgis-runtime-sdk-for-net-1006-and-preview-of-support-for-wpf-for-net-core-30. Consultado en 14 de febrero de 2022

[14] **Aspose** official site. https://products.aspose.com/gis/net. Consultado en 14 de febrero de 2022

[15] **Map Tile** Github Docs. https://github.com/apdevelop/map-tile-service-asp-net-core/blob/master/README.md. Consultado en 14 de febrero de 2022

[16] Cesar de la Torre Llorente, Unai Zorrilla Castro, Javier Calvarro Nelso, Miguel Angel Ramos *Guia de Arquitectura N-Capas Orientada al Dominio* (1ra edición), 2010 Microsoft Ibérica S.R.L

[17] MongoDB Official Site. URL: https://mongodb.com.es. Consultado en 14 de febrero de 2022.

[18] Mora Linet. OpenLatinoServer 2.0.0: Herramientas para la mejora en el rendimiento de la generación de imágenes. Habana : Matcom, 2016. Consultado en 14 de febrero de 2022.

[19] Postman Official Site - Learning Url: https://learning.postman.com. Consultado en 14 de febrero de 2022.

[20] Alan A.A. Donovan, *Go Programming Language*, 1st Edition 2015 Addison-Wesley Professional Computing Series

[21] GitHub Gobench Docs Url: https://github.com/cmpxchg16/gobench. Consultado en 14 de febrero de 2022.

[22] PostgreSQL Official Site Url: https://postgresql.org. Consultado en 14 de febrero de 2022.

[23] Daher L. Mora Valdés, *Proveedor de datos Shapefile y Consultas Espaciales en OpenLatinoServer.*, Habana : Matcom, 2019. Consultado en 14 de febrero de 2022.

[24] Carlos E. Aguilera Medina, *Adición de herramientas para la gestión de información geográfica en el visor de mapas OpenLatinoViewer: ?Get−FeatureInfo?, ?Spatial−Query?, ?Advanced-Query?, ?View−Focus−Highlight? y ?Legend?.*, Habana : Matcom, 2019. Consultado en 14 de febrero de 2022.

[25] Hirán Díaz de Acevedo Guanche , *Autenticación, Autorización y Generación de Mapas Temáticos en OpenLatino Server*, Habana : Matcom, 2019. Consultado en 14 de febrero de 2022.

[26] Frankie Mujica Cal, *Desarrollo de la nueva versión multiplataforma de OpenLatino Server*, Habana: Matcom, 2020. Consultado en 14 de febrero de 2022

[27] Frankie Mujica Cal, *Manual de Usuario de OpenLatino Server V2*, Habana: Matcom, 2020. Consultado en 14 de febrero de 2022