Ciencias Matemáticas, Vol. 35, No. 2, 2021, Pag. 111-122
Recibido 04-2021, Aceptado 08-2021

OpenLatinoMurciaClient: SIG Web con tecnologías abiertas
OpenLatinoMurciaClient: Web GIS with open technologies

Aracelys García Armenteros[1]*, Joanna Campbell Amos[1], Vivian Plasencia Calaña[2]

Resumen En la sociedad actual resulta muy frecuente que organizaciones o personas accedan a internet para consultar o crear mapas que comunican, analizan, comparten y resuelven problemas complejos alrededor del mundo. Un SIG Web es un sistema de información distribuida que comprende al menos un servidor y un cliente, donde el servidor es un servidor SIG (Sistema de Información Geográfica) y el cliente es un buscador web, aplicación de escritorio, o aplicación móvil. En el presente trabajo se desarrolla el SIG Web OpenLatinoMurciaClient (OLMC) empleando herramientas modernas y de código abierto, así como buenas prácticas de programación. En OLMC se incluye y se adapta el visor de mapas OpenLatinoView y se utiliza el servidor de mapas OpenLatinoServer, se implementa un módulo de administración de usuarios y se agregan nuevas herramientas al visor de mapas. El resultado que se presenta es un software funcional, robusto y extensible que cumple con todos los requerimientos expuestos por el cliente.

Abstract In today's society, it is very common for organizations or people to access the internet to consult or create maps that communicate, analyze, share and solve complex problems around the world. A Web GIS is a distributed information system that comprises at least one server and one client, where the server is a GIS (Geographic Information System) server and the client is a web browser, desktop application, or mobile application. In the present work the Web GIS OpenLatinoMurciaClient (OLMC) is developed using modern and open source tools, as well as good programming practices. OLMC includes and adapts the OpenLatinoView map viewer and uses the OpenLatinoServer map server. The user administration module is also implemented and new tools are added to the map viewer. The result is a functional, robust and extensible software that guarantees the specifications required by the client.

Palabras Clave
SIG Web, OpenLatinoView, OpenLatinoServer

Keywords
Web GIS, OpenLatinoView, OpenLatinoServer

[1] *Departamento de Programación e Ingeniaría de software, Universidad de la Habana, La Habana, Cuba, aracelys@matcom.uh.cu, joanna@matcom.uh.cu*
[2] *Trabajador por cuenta propia, v.pcalana@gmail.com*
***Autor para Correspondencia, Corresponding Author**

Introducción

Los avances tecnológicos y el hecho de que buena parte de las actividades humanas tengan un componente locacional, provocan que en casi cualquier ámbito se puedan mejorar los procesos desarrollados añadiendo Sistemas de Información Geográfica (SIG).

Los SIG son herramientas que nos permiten trabajar con bases de datos y realizar análisis multicriterio para la toma de decisiones. Tienen aplicación en diversos campos como son urbanismo, gestión de patrimonio cultural, redes de tensión eléctrica, cableado telefónico, gestión de rutas de transporte, edificación y obras civiles, arquitectura y turismo.

Un Sistema de Información Geográfica es un conjunto de herramientas que integra y relaciona diversos componentes (usuarios, hardware, software y procesos) que permiten la organización, el almacenamiento, la manipulación, el análisis y la modelación de grandes cantidades de datos procedentes del mundo real. Dichos datos están vinculados a una referencia espacial, esto facilita la incorporación de aspectos sociales-culturales, económicos y ambientales a los mismos con el objetivo de hacer más eficiente la toma de decisiones [1].

El SIG funciona como una base de datos con información geográfica en la que los objetos (provenientes de mapas digitales) que se encuentran almacenados en ella se asocian entre

sí a partir de un identificador común (establece una geo referencia). De esta forma, dado cualquier objeto almacenado en la base de datos, podemos conocer su localización geográfica e, inversamente, dada una localización geográfica podemos conocer todos los objetos y atributos geográficos asociados a ella [2].

Los SIG almacenan información procedente del mundo real como pueden ser, por ejemplo, la referida a carreteras, ríos, lagos, montañas, etc. Existen dos formas principales en las que se representan y se almacenan los datos antes mencionados [3]:

- raster o de retícula.
- vectorial.

Con el uso frecuente de internet, la forma de distribuir la información cambió y surgieron los SIG Web. Desde ese momento, no solo se podía acceder a las funcionalidades de un SIG desde una misma computadora, sino desde cualquier computadora o dispositivo con un navegador Web y acceso a internet.

SIG Web es un tipo de sistema de información distribuida que comprende al menos un servidor y un cliente, donde el servidor es un servidor SIG y el cliente es un buscador web, aplicación de escritorio, o aplicación móvil [4, 5].

Un SIG Web puede ser usado por varios usuarios a la vez. Esto permite que en una organización cada miembro no se tenga que instalar el software en su computadora personal y todos puedan acceder al SIG simultáneamente mediante la Web. Dos componentes fundamentales en un SIG Web son:

- Visor de mapas: Aplicación web pensada para la visualización y la consulta de información geográfica haciendo uso de los estándares de la OGC [6].
- Servidor de mapas (en inglés IMS: Internet Map Server): Provee cartografía a través de la red tanto en modo vectorial como con imágenes. La especificación estándar para estos servidores es la OGC WMS (Open Geospatial Consortium Web Map Service) [7].

A pesar de existir varias diferencias entre los SIG y los SIG Web, no se puede determinar que uno sea mejor que el otro, dependiendo de las necesidades del usuario se debe elegir cuál resulta más adecuada. Algunos aspectos que hacen diferente un SIG Web de un SIG son [4]:

- Usuarios no necesariamente expertos en SIG: Los SIG requieren de usuarios expertos en el tema mientras que un SIG Web busca tener una interfaz amigable y ser usado por usuarios no expertos en SIG.
- Alcance global: Los SIG se instalan en una computadora personal y solo puede acceder un usuario, mientras que los SIG Web están en la Web y pueden ser accedidos por varios usuarios a través de diversos equipos como computadoras personales, teléfonos móviles, tabletas electrónicas y computadoras personales portátiles.
- Gran cantidad de usuarios simultáneamente: Generalmente cuando se trabaja con un SIG es un solo usuario a la vez, con la llegada de los SIG Web eso cambia, pues muchos usuarios pueden trabajar simultáneamente con la misma herramienta.
- No está limitado a un sistema operativo específico: Los SIG para funcionar, dependen del sistema operativo que tenga la computadora donde se instale el software. Los SIG Web no dependen del sistema operativo que tenga el dispositivo desde el cual se va a utilizar porque son aplicaciones web, se puede acceder a ellos mediante navegadores web de Internet como: Google Chrome, Mozilla Firefox, Opera, Safari e Internet Explorer.

1. Motivación, justificación y formulación del problema

En el marco del proyecto Cadic-UH desarrollado por la Casa del Software, que pertenece a la facultad de Matemática y Computación de la Universidad de La Habana, se generaron una serie de requerimientos funcionales y de entorno muy específicos por parte de los clientes, lo cual constituye la motivación para la realización de este trabajo. Los requerimientos son los siguientes:

1. Guardar la imagen del mapa que se está observando en ese momento.
2. Editar la imagen del mapa que se está observando en ese momento. Permitir guardarla e insertar nuevas imágenes.
3. Descargar en distintos formatos vectoriales (WKT, GeoJSON) las geometrías que estén seleccionadas mediante cercados en el mapa.
4. Añadir geometrías al mapa. Las geometrías pueden ser añadidas en distintos formatos.
5. Trabajar con la visibilidad de las capas. Las capas se deben poder mostrar, esconder y modificar la opacidad.
6. Gestionar usuarios. Se puedan crear, mostrar, actualizar y borrar usuarios. Además, los usuarios deben tener restricciones de acceso a las capas y funcionalidades del mapa.
7. Permitir la conexión a la fuente de servicio WMS6.
8. Cuando un elemento es seleccionado debe poder mostrarse la información que tiene asociada.
9. Herramientas que permitan al usuario medir distancias y áreas. Las líneas o superficies dibujadas por el usuario también deben incluirse en la edición del mapa, de forma que se pueda utilizar esa misma herramienta para remarcar una zona o elemento.
10. Permitir distintos tipos de búsquedas sobre el mapa.
11. Permitir descargar en distintos formatos vectoriales (WKT, GeoJSON) las geometrías resultado de las búsquedas realizadas.
12. Usar tecnologías de código abierto y de amplio uso.
13. El software tiene que ser multiplataforma. Debe poder funcionar en Windows y Linux.

En la fase inicial de la investigación se estudiaron varios SIG Web que se utilizan en la actualidad con el objetivo de establecer semejanzas y diferencias en sus funcionalidades respecto a los requerimientos solicitados por el cliente. Entre los SIG Web analizados se encuentra OpenStreetMap [8], Fluvanna County, VA Geographic Information System [9], Abbotsford [10] y Scotland's environment [11].

Una vez analizados estos SIG Web se identificaron funcionalidades que son iguales o semejantes a las requeridas, pero algunos no son de código libre o presentan problemas de compatibilidad, por lo que no resulta factible la utilización de ninguno de ellos y se determina realizar un SIG Web tomando ideas para el diseño de la interfaz visual en cuanto a:

- La ubicación de los botones de las funcionalidades del visor de mapas dentro del visor de mapas.
- Mostrar en la herramienta LayerSwitcher las capas que están activas y las que no.
- La ubicación de la información personal de un usuario luego de iniciar sesión en la esquina derecha superior de la aplicación.

En la Casa del Software se tiene el código fuente del visor de mapas OpenLatinoViewer (OLV) [12], desarrollado en este mismo grupo, que ya tiene implementado algunos requerimientos y se pueden obtener varias funcionalidades que se necesitan ampliando las ya existentes. Se posee además el servidor de mapas OpenLatinoServer (OLS) [23], igualmente desarrollado en este grupo, que puede ser utilizado como servidor de mapas para el SIG Web que se desarrollará.

Teniendo esto en cuenta, los requerimientos funcionales y de entorno que se deben incorporar son:

1. Guardar la imagen del mapa que se está observando en ese momento.
2. Editar la imagen del mapa que se está observando en ese momento. Permitir guardarla e insertar nuevas imágenes.
3. Descargar en distintos formatos vectoriales (WKT, GeoJSON) las geometrías que estén seleccionadas mediante cercados en el mapa.
4. Añadir geometrías al mapa. Las geometrías pueden ser añadidas en distintos formatos.
5. Trabajar con la visibilidad de las capas. Las capas se deben poder mostrar, esconder y modificar la opacidad.
6. Gestionar usuarios. Se puedan crear, editar, actualizar y borrar usuarios. Además, los usuarios deben tener restricciones de acceso a las capas y funcionalidades del mapa.
7. Permitir descargar en distintos formatos vectoriales (WKT, GeoJSON) las geometrías resultado de las búsquedas realizadas.
8. Manual que sirva a los desarrolladores para poder comprender la funcionalidad y facilitar el mantenimiento y posibles mejoras posteriores.

Analizando el contexto de la investigación, se puede definir el problema identificado de la siguiente forma: aunque en la actualidad existen varios SIG Web, no se encontró ninguno que tuviese implementados todos los requerimientos solicitados, o fuera de código abierto y compatible con las herramientas y tecnologías utilizadas en la Casa del Software.

1.1 Objetivos

Para darle solución al problema planteado, el objetivo general del presente trabajo es desarrollar el SIG Web OpenLatinoMurciaClient (OLMC), usando buenas prácticas de programación, que cumpla con todos los requerimientos planteados, sea extensible y tenga integrado el visor de mapas OLV y el servidor de mapas OLS.

Para garantizar el cumplimiento del objetivo general planteado, se propone alcanzar, paulatinamente, un conjunto de objetivos específicos. Son los siguientes:

1. Diseñar una nueva aplicación llamada OpenLatinoMurciaClient con tecnologías que sean modernas y de código abierto.
2. Diseñar e implementar una arquitectura extensible que integre OLMC con el visor OLV y el servidor OLS.
3. Implementar un módulo de administración en OLMC que permita administrar usuarios.
4. Extender la herramienta View-Focus-Highlight de OLV, para permitir que se puedan descargar las geometrías resultantes en distintos tipos de formatos vectoriales (WKT, GeoJSON).
5. Implementar las siguientes nuevas herramientas en el visor de mapas:
   - SaveMapImage: Herramienta para descargar una foto del mapa.
   - LayoutMaker: Herramienta para editar la foto del mapa. Posibilita guardar la imagen que se está editando y cargar nuevas imágenes.
   - DownloadGeometries: Herramienta para descargar en distintos formatos geometrías que estén en el área seleccionada.
   - UploadGeometries: Herramienta para adicionar en distintos formatos y mostrar geometrías en el mapa.
   - LayerSwitcher: Herramienta que permite trabajar con la visibilidad de las capas que posee el mapa.

2. Diseño de la aplicación OLMC

OpenLatinoMurciaClient (OMLC) es la propuesta de este trabajo para dar solución al problema planteado anteriormente. A continuación se presenta la estructura de la aplicación OLMC a través de una gráfica y las acciones que pueden realizar los usuarios en la aplicación. Además se explica cómo se integraron el visor OLV y el servidor OLS en la aplicación creada y se exponen las nuevas herramientas creadas en el visor de mapas.

2.1 Funcionalidades

En la aplicación existen dos roles: Usuario Registrado y Admin.

1. Un Usuario Registrado es un usuario que ha sido insertado en la base de datos del sistema para que tenga acceso a las funcionalidades de la aplicación.

2. Un Admin es un administrador del sistema. Tiene permisos para realizar las operaciones Crear, Leer, Actualizar y Borrar de los usuarios. Registra usuarios en el sistema. Está registrado en el sistema y puede acceder a todas las funcionalidades del visor de mapas.

Un Workspace, como su nombre en inglés indica, es un espacio de trabajo. Ese concepto existe en la aplicación para cada Usuario Registrado y permite configurar los permisos de acceso a las capas de la cartografía y funcionalidades del visor de mapas. Cada usuario del sistema tiene uno o más Workspaces asignados.

Un Usuario Registrado, con todos los permisos posibles permitidos, puede acceder a las funcionalidades del visor de mapas OLV que se muestran en la Figura 1.Las funcionalidades 1, 2, 3, 4, 5, 6 y 7 ya existían en el visor de mapas [12]. La funcionalidad número 8 se modificó para ser extendida; anteriormente solo se permitía descargar los resultados de una consulta en formato CSV y la herramienta no era extensible, entonces, se diseñó e implementó una herramienta extensible que permite descargar además de CSV, otros formatos como WKT y GeoJSON. Las funcionalidades 9, 10, 11, 12 y 13 se implementan para cumplir con los requerimientos de la aplicación planteados por el cliente.

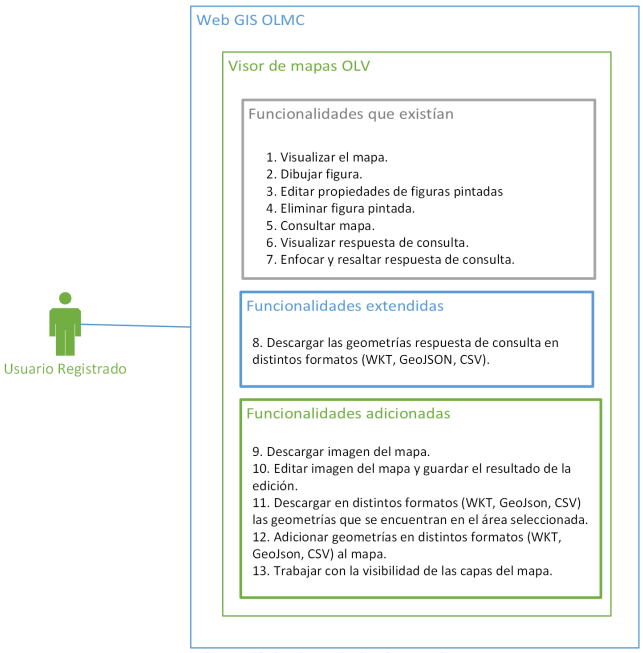

Figura 1. Funcionalidades del visor de mapas OLV

2.2 Arquitectura de software

Luego de estudiar varias arquitecturas de software, se implementa Clean Architecture (Arquitectura Limpia, en idioma español) porque se necesita que la aplicación sea extensible para adicionar funcionalidades en el futuro [13, 14]. Esta arquitectura permite mantener el código desacoplado lo que facilita la extensión y las pruebas al software. En la Figura 2 se muestra la Arquitectura de software Clean Architecture implementada en OLMC. Se puede observar la separación de las capas y la dependencia que existe entre ellas. Web.Api.Core representa el círculo más interno y Web.Api.App, el más externo, según el diagrama oficial. Cumple con la Regla de Dependencia de Clean Architecture pues ninguna capa depende de otra externa a ella. Posee tres capas:

1. Web.Api.Core: Es la capa base. No tiene dependencias externas. Posee las entidades y la declaración de las interfaces.

2. Web.Api.Infrastructure: Depende de la capa Web.Api.Core. Tiene la implementación de las interfaces que se declaraon en Web.Api.Core. Además, posee el Seed (clase que inserta inicialmente datos específicos en la base de datos), las migraciones y el contexto de la base de datos.

3. Web.Api.App: Depende de las capas Web.Api.Infrastructure y Web.Api.Core. Tiene el Sistema de Gestión de Base de Datos (SGBD) utilizado y los controladores del servidor de la aplicación.

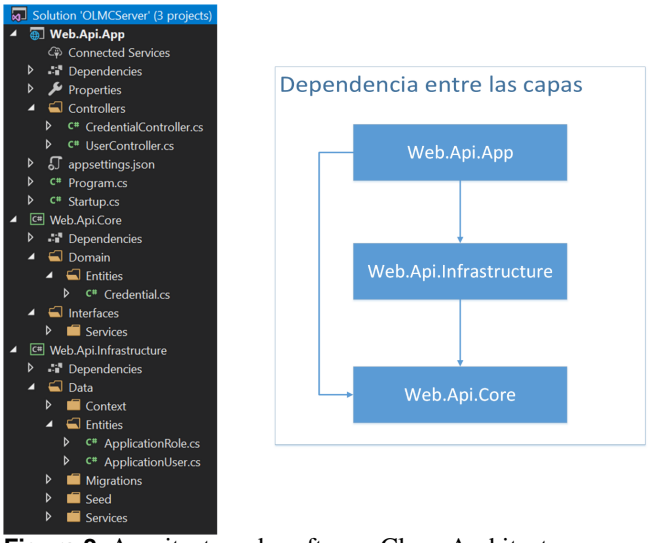

Figura 2. Arquitectura de software Clean Architecture implementada en OLMC

Para cumplir con Clean Architecture, todas las entidades de la base de datos deben estar en Web.Api.Core y esta capa no debe referenciar a ninguna librería ni capa externa. Con la utilización de Asp.Net Core Identity para crear usuarios y roles, se decide poner las entidades relacionadas con Asp.Net

Core Identity en la segunda capa, Web.Api.Infrastructure, referenciando todo de Identity en esa capa. De esta forma se incumple el principio de Clean Architecture de que todas las entidades de la base de datos deben estar en la capa más interna y se cumple que la capa más interna Web.Api.Core, no referencia a ninguna librería ni capa externa. Como se puede observar en la Figura 2 se elige esta variante para mantener el principio de que la capa más interna no depende de ninguna librería ni capa externa. Si se adiciona una nueva entidad, el programador debe conocer Clean Architecture y entender cómo está implementada en el servidor de OLMC, la adicionará en la capa Web.Api.Core si no depende de una librería externa.

2.3 Patrones

Modelo-Vista-Controlador (MVC) es un patrón de diseño usado para desacoplar user-interface (vista), datos (modelo) y la lógica de la aplicación (controlador). Este patrón ayuda a lograr la separación de preocupaciones [15, 16].

En la Figura 3 se puede observar el funcionamiento del patrón MVC en la aplicación OLMC. Cuando un usuario realiza un pedido ocurre lo siguiente [22]:

1. El usuario realiza una acción en la UI de OLMC y se forma un pedido que según la acción que realizó, es enviado hacia el servidor de OLMC o el de mapas OLV.
2. El servidor recibe el pedido en un controlador que organiza la información y la envía al modelo.
3. El modelo, resuelve el pedido con la información que le llegó del controlador y envía al controlador los datos que se necesitan para que el usuario obtenga la respuesta que desea.
4. El controlador envía a la vista la información que le llegó del modelo.
5. La vista muestra la respuesta al pedido realizado por el usuario.

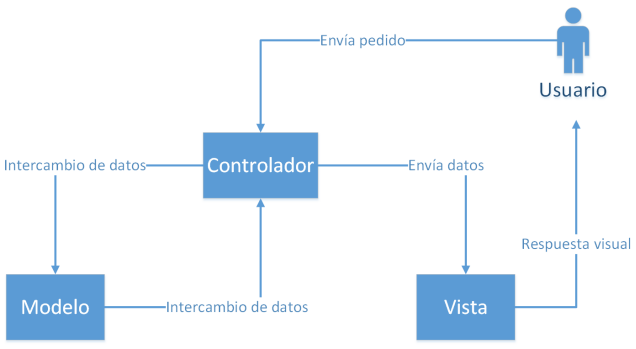

Figura 3. Patrón MVC en OLMC

El patrón Repositorio consiste en separar la lógica que recupera los datos y los asigna a un modelo de entidad de la lógica de negocios que actúa sobre el modelo, esto permite que la lógica de negocios sea independiente del tipo de dato que comprende la capa de origen de datos, en pocas palabras un repositorio, media entre el dominio las capas de mapeo de datos. El patrón Unidad de Trabajo, es la forma correcta de implementación del patrón Repositorio. Este centraliza las conexiones a la base de datos realizando un seguimiento de todo lo que sucede durante una transacción cuando se usan las capas de datos y revertirá la transacción si Commit() no se ha invocado o existen incongruencias lógicas [17]. OLMC tiene los patrones Repositorio y Unidad de Trabajo [18, 19, 22].

2.4 Modelo de datos

Para almacenar los datos necesarios para el cumplimiento de los requerimientos del cliente, es fundamental el uso de una base de datos. Se realiza el diseño del modelo de datos para q la base de datos sea extensible. Las tablas que contienen en el nombre el prefijo AspNet, son creadas por Asp.Net Core Identity. Como la aplicación debe poseer usuarios y permitir la restricción de estos a distintas capas y funcionalidades del visor de mapas, se generaron usando Identity, las tablas de la base de datos: AspNetUser y AspNetRole. En AspNetUser se almacena toda la información de los usuarios y en AspNetRole, la de los Workspaces que existen en la aplicación. Como se puede observar en la Figura 4, un usuario puede tener cero o más Workspaces y un Workspace puede formar parte de cero o más usuarios. La tabla AspNetUser de Identity se extendió. Se le agregó la columna isAdmin para saber si un usuario es Administrador de la aplicación o no. Los datos almacenados son de tipo bool. El campo isAdmin tiene valor True si el usuario es Admin y False si no lo es. La tabla AspNetRole también fue extendida, se le adicionó la columna ServerId para guardar el Id que posee el Workspace de esa fila en la base de datos del servidor OLS. La tabla Credential se crea para almacenar la información del servidor OLS. Esta tabla no se relaciona con ninguna otra. Los campos que posee son:

1. Id: Identificador del servidor.
2. ServerName: Nombre del servidor.
3. AccessToken: Token de acceso al servidor.
4. UpdateKey: Llave de actualización.
5. ServerUrl: Url que se utiliza para acceder al servidor.

3. Implementación de herramientas de OLMC

Después de realizar un estudio sobre las tecnologías y herramientas a emplear para la implementación del SIG Web OLMC [22] se determinó utlizar C# como lenguaje de programación en el servidor, ASP.NET 2.2 Core es el framework de desarrollo web utilizado y Entity Framework Core 2.2 es la tecnología utilizada para el acceso a los datos en el servidor. El lenguaje de programación utilizado en la parte del cliente

es TypeScript 3.7.5, para desarrollar el front-end de la aplicación se utiliza el framework Angular en su versión 9.0.2 y OpenLayers 4.6.0 es la librería que se utiliza para mostrar el mapa en la aplicación. Se utilizó como Integrated Development Environment (IDE) para su desarrollo Microsoft Visual Studio 2017 Community Edition y SQL Server como gestor de base de datos.

A continuación se explican detalles de implementación para entender el funcionamiento de algunas herramientas y cómo se extendieron. Para obtener un código fácil de refactorizar y adaptar se siguen los principios SOLID [20, 21] de la programación orientada a objetos, los cuales permiten que los proyectos sean más legibles, mantenibles, robustos y escalables en el futuro.

3.1 Implementación de la herramienta LayerSwitcher

Para que una capa del mapa forme parte del LayerSwitcher, es necesario que, al construirse la capa, la propiedad que se llama displayInLayerSwitcher esté en true. En la construcción de las capas del mapa que son pedidas al servidor de mapas OLS cada capa imageLayer que se crea nueva no posee la propiedad displayInLayerSwitcher, entonces se toma el valor por defecto de esta propiedad que es true, por tanto, esas capas se muestran en el LayerSwitcher.

3.2 Integración de OLMC con el servidor OLS

El servidor OLS tiene implementado el protocolo de seguridad JSON Web Token (JWT) . Por esta razón, cada pedido que se le hace a OLS, debe cumplir con una estructura específica. Los pasos que se mencionan a continuación se tienen que seguir para realizar un pedido al servidor OLS:

1. Pedir al servidor de OLMC el token para acceder a OLS.
2. Si el token no ha expirado realizar el pedido que se quiere y saltarse los restantes pasos.
3. Si el token ya expiró, pedir la llave de actualización que está en el servidor de OLMC.
4. Con el token de acceso y la llave de actualización, mandar el pedido al servidor OLS para actualizar el token de acceso y la llave de actualización, para poder acceder luego a las demás funcionalidades de OLS. Ese pedido devuelve un nuevo token de acceso y una nueva llave de actualización.
5. Guardar la nueva llave y el nuevo token de actualización en la base de datos de OLMC.
6. Con el nuevo token realizar el pedido que se quiere.

En OLMC existe una función por la que pasan todos los pedidos realizados al servidor de mapas. Cada vez que se quiera realizar un pedido, se llama a esa función para no tener que implementar todos los pasos anteriormente mencionados [22].

3.3 Implementación de la herramienta DownloadGeometriesFormats

La herramienta DownloadGeometriesFormats es la encargada de descargar geometrías en distintos formatos. El resultado siempre es un archivo zip que contiene un fichero (con extensión del formato de geometría seleccionado) por cada capa a descargar con las correspondientes geometrías. El formato de las geometrías guardadas en el servidor OLS es WKT. Todos los formatos en los que se quiera descargar las geometrías se construyen a partir de WKT u otro formato de descarga anteriormente implementado. La herramienta DownloadGeometriesFormats se ha implementado como una componente individual, de esta forma puede ser llamada desde otras herramientas que quieran descargar geometrías. Dos herramientas que la utilizan son: DownloadGeometries y View-Focus-Highlight.

3.3.1 Herramienta DownloadGeometries

La herramienta DownloadGeometries es la encargada de descargar en distintos formatos las geometrías de un área seleccionada. El resultado es un archivo zip que contiene un fichero (con extensión del formato de geometría seleccionado) por cada capa con las geometrías que se encuentran en el área seleccionada. Esta área está dada siguiendo la forma de cercado que existía ya en el visor de mapas OLV con la herramienta de consulta Spatial-Query [12]. Esta herramienta llama a la herramienta DownloadGeometriesFormats para descargar las geometrías.

3.3.2 Extensión de la herramienta View-Focus-Highlight

En OLV existía ya una herramienta llamada View-Focus-Highlight. Es la encargada de la visualización de todas las consultas realizadas en el visor de mapas [12]. Esta se ha extendido porque anteriormente permitía descargar las geometrías resultado de las consultas solamente en formato CSV. Ahora permite descargar las geometrías en distintos formatos, llamando a la herramienta DownloadGeometriesFormats

3.4 Implementación de la herramienta UploadGeometries

UploadGeometries es la herramienta que permite adicionar geometrías en distintos formatos en el visor de mapas. En el visor de mapas las geometrías se pintan con OpenLayers, librería que soporta ciertos formatos para pintar geometrías y son los únicos que se pueden usar. Si se quiere pintar geometrías en formatos que OpenLayers no soporta, entonces habrá que transformar esas geometrías a un formato soportado por OpenLayers y luego mandarlas a pintar. Las geometrías se pintan siempre sobre una nueva capa que es nombrada por el usuario y se muestra en el LayerSwitcher. En el código esa nueva capa se llama vectorLayer.

4. Validación del cumplimiento de los objetivos

Para validar el correcto funcionamiento de la las herramientas implementadas se realizaron pruebas de funcionalidad

[22]. Las pruebas se realizaron en una laptop con Procesador Intel Core i7 y Memoria RAM 4 GB. A continuación se presentan los resultados de los principales escenarios de prueba realizados.

Las pruebas de funcionalidad iniciaron con los siguientes elementos:

- Tres capas en el mapa: points. Se pinta en el mapa de color rojo. buildings. Se pinta en el mapa de color verde. roads. Se pinta en el mapa de color negro.
- 3 Workspaces: Workspace0: Tiene acceso a las tres capas del mapa. No tiene acceso a las funcionalidades Spatial-Query, Advanced-Query y DownloadGeometries. Workspace1: Tiene acceso a las capas points y buildings. Tiene acceso a todas las funcionalidades del mapa. WorkspaceAll: Tiene acceso a las tres capas. Tiene acceso a todas las funcionalidades.
- 2 usuarios creados: Email: adminolmc@gmail.com. Name: AdminOLMC. Workspaces: WorkspaceAll. Es admin de la aplicación. Email: user0@gmail.com. Name: User0. Workspaces: Workspace0, Workspace1. No es Admin de la aplicación.

Todas las señalizaciones en las imágenes se realizaron con el color anaranjado. La Figura 5 muestra las funcionalidades del visor de mapas que se pueden habilitar o deshabilitar, según el Workspace que se esté utilizando. Estas son las únicas que dependen del servidor de mapas. Las demás funcionalidades siempre pueden ser usadas sin importar el Workspace.

4.1 Funcionamiento de los Workspaces, restricción en las funcionalidades y capas en el visor de mapas

La Figura 6 muestra el usuario User0 está utilizando el Workspace Workspace1. El visor de mapas posee todas las funcionalidades y tiene acceso a todas las capas excepto roads, demostrando el correcto funcionamiento del inicio de sesión de los usuarios y la herramienta LayerSwitcher.

4.2 Herramientas SaveMapImage y LayoutMaker

La herramienta SaveMapImage descarga la imagen que se muestra en el visor de mapas, mientras LayoutMaker se utiliza para editar la imagen mostrada en el visor de mapas. En la Figura 7 se muestra el modal al hacer clic izquierdo en la herramienta SaveMapImage.

En la Figura 8 se puede observar el modal que se muestra al seleccionar la herramienta LayoutMaker. Desde esta herramienta con las diversas prestaciones que tiene, se puede editar la foto mostrada en el visor de mapas y se pueden guardar los cambios realizados.

4.3 Descargar geometrías y cargar geometrías en distintos formatos en el visor de mapas.

Para mostrar el correcto de las herramientas DownloadGeometriesFormats y UploadGeometries se realiza el cercado del área para descargar las geometrías de cada capa del visor de mapas. Comprende lo que está dentro y sobre el círculo pintado. Luego, se selecciona la herramienta DownloadGeometries del visor de mapas, aparece un modal para poner el nombre que se quiere que tenga el archivo zip en el que se guardarán las geometrías y el formato de las mismas como se muestra en la Figura 9.

5. Conclusiones

Con la realización de este trabajo se puede concluir que se cumplieron los objetivos planteados inicialmente. Se cuenta con un software funcional, robusto y extensible que cumple con todos los requerimientos expuestos por el cliente. Se hizo un estudio del estado del arte de los SIG Web y se tomaron ideas en cuanto al diseño de la interfaz visual, específicamente la ubicación de las funcionalidades del visor de mapas con botones dentro del mismo, las forma de activar y desactivar capas en el LayerSwitcher con checkbox y la ubicación de la información personal del usuario cuando inicia sesión en la esquina superior derecha. La implementación de OLMC se realizó usando herramientas modernas y de código abierto, siguiendo la arquitectura de software Clean Architecture y patrones de diseño como MVC y Repository Pattern con Unit of Work. En OLMC se incluyó y adaptó el visor de mapas OLV y se utiliza el servidor de mapas OLS. Se implementó un módulo de administración de usuarios. Se agregaron nuevas herramientas al visor de mapas como son: SaveMapImage, LayoutMaker, DownloadGeometries, UploadGeometries y LayerSwitcher. Algunos otros elementos también fueron agregados al visor de mapas para facilitar su uso como: Escala del mapa, Latitud y Longitud del mapa y FullScreen.

5.1 Líneas futuras de investigación

Aunque se cumplieron todos los objetivos planteados al inicio del trabajo, el software implementado puede mejorarse y se pueden agregar nuevas funcionalidades, por ello, se propone:

1. Extender la herramienta UploadGeometries y permitir añadir geometrías al seleccionar un archivo que se encuentre en la computadora personal y tenga en su contenido, las geometrías. Permitir añadir geometrías arrastrando el archivo que contiene las geometrías en el visor de mapas.
2. Implementar nuevos formatos de geometrías para trabajar en el visor de mapas, tanto para descargar como para añadir. Ejemplo de nuevos formatos pueden ser: Shapefile, TopoJSON y KML.
3. Implementar en el visor de mapas una herramienta para la edición de capas. Esto se puede hacer con facilidades que brinda OpenLayers.
4. Implementar en el visor de mapas una herramienta para imprimir mapas a distintas escalas (predefinidas) y

distintos tamaños de papel. Añadir esta herramienta en LayoutMaker también.

5. Añadir nuevos tipos de búsqueda en el visor de mapas.

6. Añadir en el visor de mapas una herramienta para ver el mapa con distintas escalas.

Referencias

[1] J. Star and J. Estes. Geographical Information Systems: An Introduction. 1990.

[2] P. Bolstad. GIS Fundamentals: A first text on Geographic Information Systems. 2005. White Bear Lake, MN: Eider Press (Second Edition).

[3] C. D. Tomlin. Geographic information systems and cartographic modeling. 1990. Prentice Hall.

[4] ArcGIS. Acerca de SIG web. URL: https://sig/AcercadeSIGweb/Documentación/ArcGISEnterprise.htm. Web; accedido 9 Agosto 2020.

[5] P. Fu. Getting to know Web GIS. 2018. Esri Press

[6] G. Andorrap. Visor de mapas. URL: https://www.cartografia.ad/visor-de-mapas. Web; accedido 9 Agosto 2020.

[7] A.Díaz. Servidores & mdash; Panorama del SIG Libre. URL: https://panorama-sig-libre.readthedocs.io/es/latest/servidores/. Web; accedido 9 Agosto 2020.

[8] OpenStreetMap. OpenStreetMap. URL: https://www.openstreetmap.org/about. Web; accedido 22 Julio 2020.

[9] V. G. I. S. Fluvanna County. Fluvanna County, VA. URL: https://www.webgis.net/va/fluvanna/. Web; accedido 23 Julio 2020.

[10] abbotsford. abbotsford. URL: https://maps.abbotsford.ca/Html5Viewer/. Web; accedido 23 Julio 2020.

[11] Scotland's environment web. S. e. web, Who are we?. URL: https://www.environment.gov.scot/about-us/who-are-we/. Web; accedido 23 Julio 2020.

[12] C. Medina. Adición de herramientas para la gestión de información geográfica en el visor de mapas OpenLatinoViewer: Get Feature Info, Spatial Query, Advanced Query, View Focus Highlight y Legend. Universidad de La Habana. Facultad de Matemática y Computación. 2019.

[13] R. Martin. Clean Architecture: A Craftsman's Guide to Software Structure and Design. 2017. Pearson.

[14] Clean Coder Blog. URL: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html. Web; accedid 2 Agosto 2020.

[15] M. Fowler. Patterns of Enterprise Application Architecture. 2002. Addison Wesley.

[16] Microsoft. ASP.NET MVC Pattern | .NET. URL: https://dotnet.microsoft.com/apps/aspnet/mvc. Web; accedido 3 Agosto 2020.

[17] E. Barrios. Patrón Repositorio (Repository Pattern) y Unidad de Trabajo (Unit Of Work) en ASP.NET Core WebApi 3.0. URL: https://dev.to/ebarrioscode/patron-repositorio-repository-pattern-y-unidad-de-trabajo-unit-of-work-en-asp-net-core-webapi-3-0-5goj. Web; accedido 4 Agosto 2020.

[18] Microsoft. Implement the infrastructure persistence layer with Entity Framework Core. URL: https://docs.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/infrastructure-persistence-layer-implementation-entity-framework-core-implement-custom-repositories-with-entity-framework-core. Web; accedido 4 Agosto 2020.

[19] S. Chauhan. Implementing Repository and Unit of Work Patterns with ASP.NET MVC. URL: https://www.dotnettricks.com/learn/mvc/implementing-repository-and-unit-of-work-patterns-with-mvc. Web; accedido 10 Agosto 2020.

[20] S. Metz. SOLID Object-Oriented Design. 2009.

[21] G. Fernández. Los 5 principios SOLID. Los principios SOLID explicados en detalle para el desarrollo de aplicaciones mantenibles y robustas. URL: https://medium.com/@ger86/los-5-principios-solid-68d697984abd. Web; accedido 410 Agosto 2020.

[22] V. Plasencia. Diseño e implementación de la aplicación Web GIS OpenLatinoMurciaClient. Universidad de La Habana. Facultad de Matemática y Computación. 2020.

[23] F. M. Cal. Desarrollo de la nueva versión Multiplataforma de OpenLatinoServer. Universidad de La Habana. Facultad de Matemática y Computación. 2020.

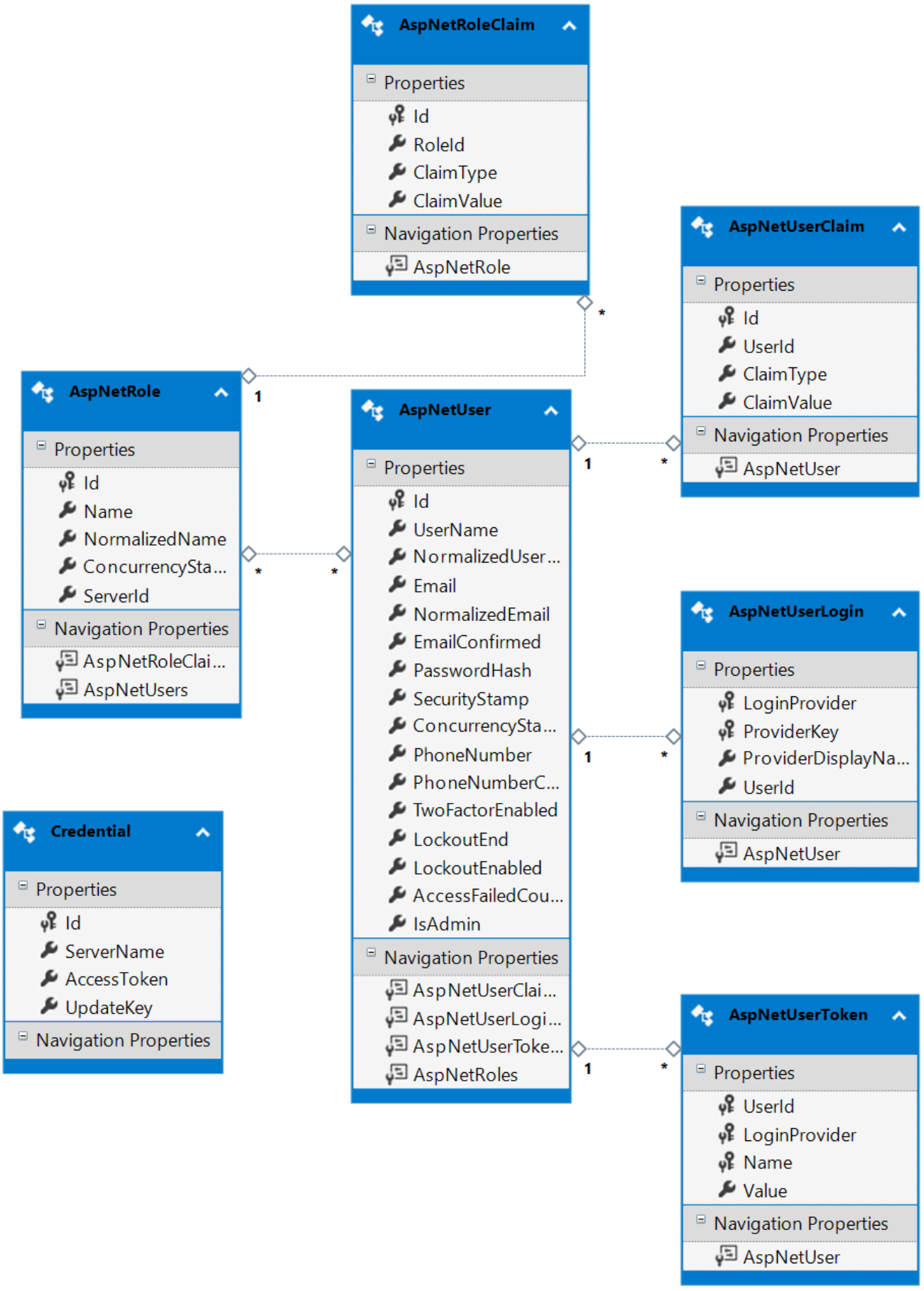

Figura 4. Modelo de datos OLMC

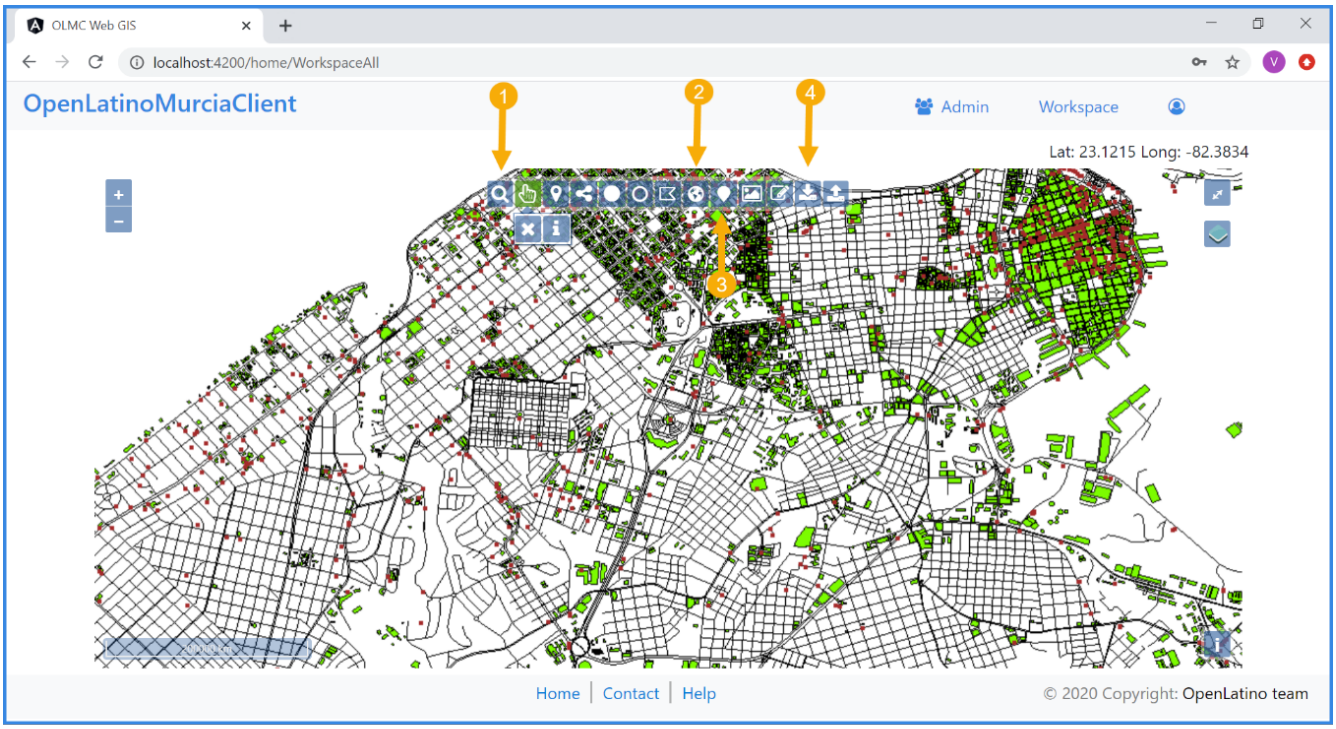

Figura 5. Funcionalidades del visor de mapas

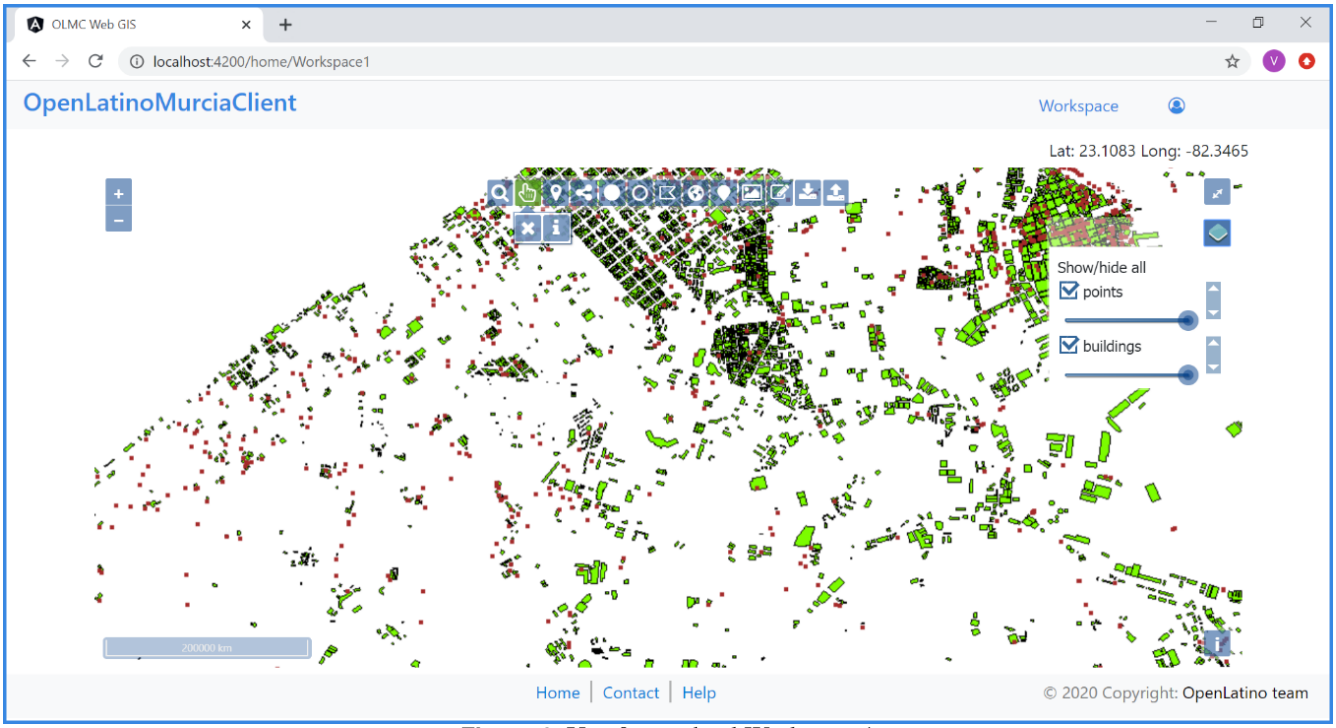

Figura 6. User0 usando el Workspace1

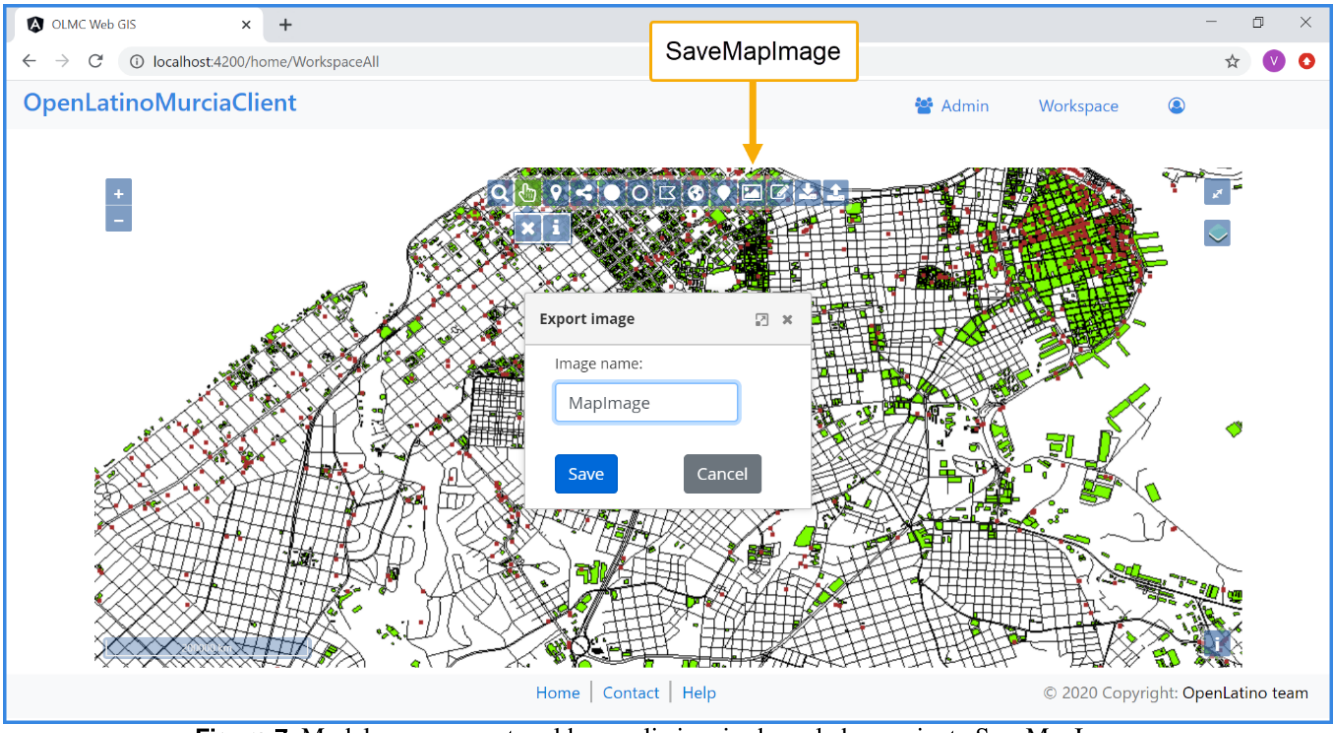

Figura 7. Modal que se muestra al hacer clic izquierdo en la herramienta SaveMapImage

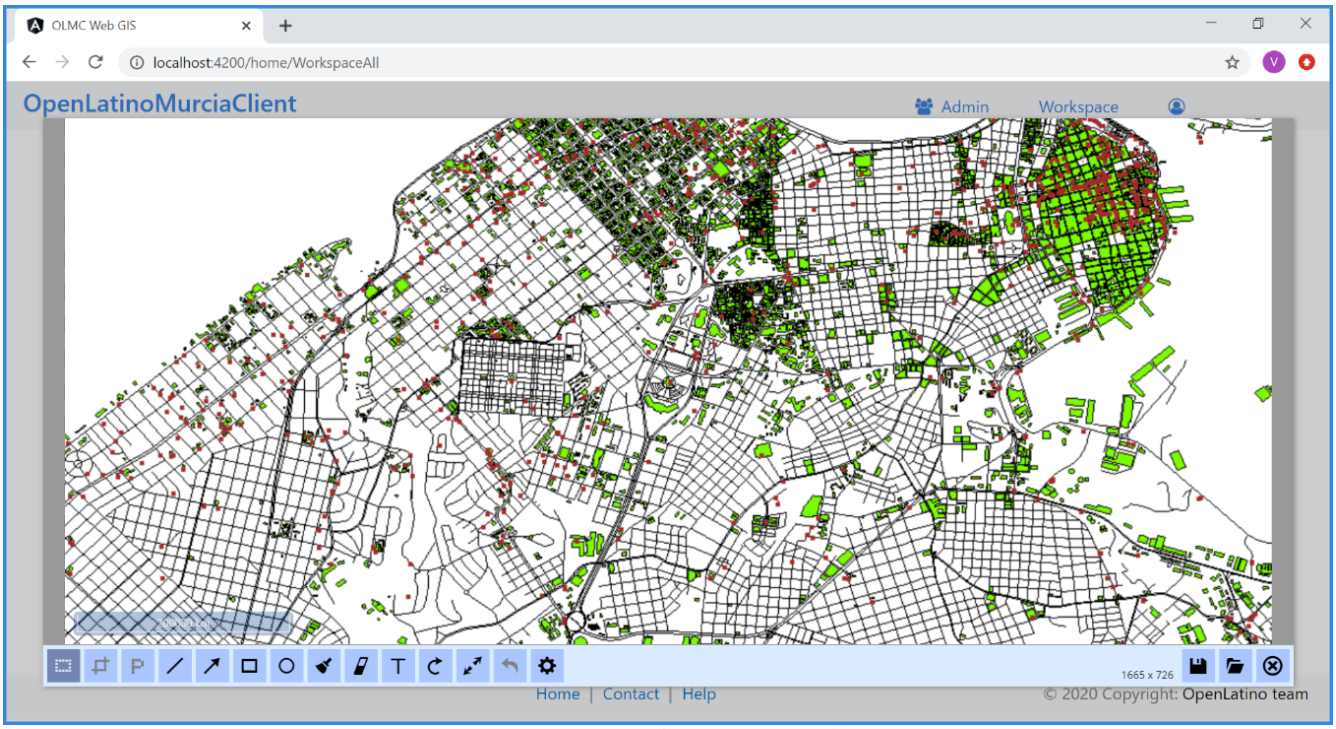

Figura 8. Herramienta LayoutMaker

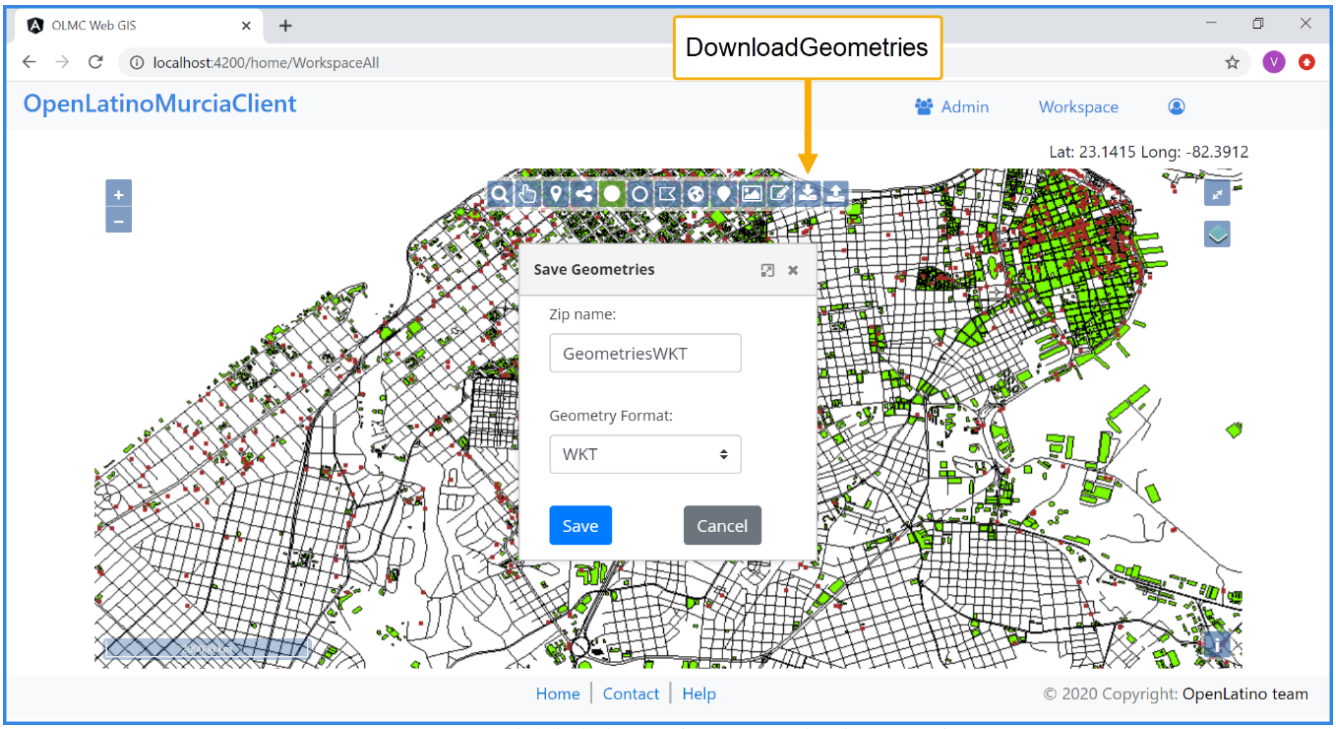

Figura 9. Modal de la herramienta DownloadGeometries