Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación

Centro de Computación Gráfica

Sistema Integrado de desarrollo para el lenguaje de sombreado de OpenGL.

Trabajo Especial de Grado
presentado ante la Ilustre
Universidad Central de Venezuela
por el bachiller

Luiyit V. Hernández G.

para optar al título de
Licenciado en Computación

Tutor
Prof. Esmitt Ramírez

Caracas, 22 de abril de 2015

Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación

Resumen

Sistema Integrado de desarrollo para el lenguaje de sombreado de OpenGL.

Autor: Luiyit V. Hernández G.
Tutor: Prof. Esmitt Ramírez.

La tecnología *shader* es cualquier unidad escrita en un lenguaje de sombreado. Los *shaders* son utilizados para producir imágenes y animaciones generadas por computador. En la actualidad, existen herramientas y tecnologías Web que permiten realizar el despliegue de este contenido directamente en el navegador Web. Estas bibliotecas están compuestas por APIs de desarrollo entre las cuales podemos mencionar WebGL, Canvas de HTML5 y ThreeJS. Este ecosistema de herramientas ha impulsado el desarrollo de soluciones que apoyan la implementación de programas de *shader* para el lenguaje de sombreado de OpenGL (GLSL). Sin embargo, de acuerdo a la investigación realizada no existen aplicaciones disponibles en la Web que tengan un criterio unificado y contemplen todos los elementos mínimos necesarios para llevar a cabo el proceso de desarrollo de *shaders*. Por ende, la programación y pruebas de *shader* en un ambiente Web se ve afectado por el insuficiente soporte con el que cuentan los procesos y tareas básicas asociadas. En este trabajo se desarrolla una solución que provee un conjunto de herramientas que permiten incorporar y enlazar los diversos elementos que intervienen al momento de desarrollar *shaders*. La solución permite a través de un conjunto de plantillas, gráficos 3D y tecnología de sincronización implementar programas GLSL. La solución fue sometida satisfactoriamente a una serie de pruebas cuantitativas y cualitativas con el objetivo de determinar el rendimiento de la tecnología utilizada, y recopilar las experiencias de los usuarios de la aplicación en términos de utilidad, facilidad de uso y cumplimiento de los objetivos de la investigación. Los resultados obtenidos indican que la solución cuenta con los módulos necesarios para ser considerada como un editor de *shaders* Web, por lo que desarrollar programas GLSL de ámbito general usando la solución desarrollada resulta factible.

Palabras clave: Editor Web, OpenGL, WebGL, *Shader*.

Agradecimientos

*Agradezco a **Dios**, por darme la sabiduría y paciencia necesaria para emprender y culminar esta extraordinaria etapa de mi vida, por permitirme contar con el apoyo de las personas correctas y estar presente en esos momentos difíciles.*

Agradezco a mi ***madre***, sin su presencia nada de esto pudo haber sido posible. Gracias por su incondicional bendición, apoyo y amor. Los principios inculcados, buenos valores y espíritu de lucha y superación me permitieron llegar hasta aquí. Siéntase orgullosa, que mi logro es suyo. Estaré eternamente agradecido.

Agradezco a mis amadas ***hermanas*** Yitlismar y Marycarmen por estar siempre a mi lado, escuchándome, ayudándome y compartiendo el día a día. De igual manera, este logro es de ustedes.

Agradezco a ***mi futura esposa*** Maddeleing por todos estos años de paciencia, amor, comprensión y presencia. Gracias por permanecer a mi lado en los momentos fáciles y no tan fáciles. Junto a mi madre y hermanas, disfruta este logro, también formas parte importante de él.

Agradezco a ***Damelis Sojo*** (amiga y consejera) por su incondicional apoyo y ayuda. Le estaré siempre agradecido por estar presente en cada momento.

Agradezco de forma especial a ***Christiam*** y ***Jhoan*** por ser parte de mi familia, y contar con su opinión y ayuda a lo largo de este camino. Gracias por todos los buenos momentos.

Agradezco a mi ***tutor*** Esmitt Ramírez, por guiarme durante la realización de este trabajo; por su paciencia, su tiempo y todas sus enseñanzas.

Agradezco a todos los ***profesores*** del Centro de Computación Gráfica, por las horas invertidas en mi formación en el área y su gran apoyo.

Agradezco a todos **mis familiares**, tíos, primos, abuelos, cuñados y ahijadas que directa o indirectamente estuvieron presentes y me brindaron su apoyo.

Agradezco a ***mis dos segundas familias***, Familia Mena y Familia Landaeta por permitirme compartir con ustedes y estar siempre dispuestos a ayudarme.

Agradezco a ***mis buenos amigos***, Edwar, Rafael, Kimelly, Cesar, Alvaro, Yudi, Jordan, Francisco, Lucas y Kelly por su apoyo en algún momento de mi carrera.

Agradezco a la ***Universidad Central de Venezuela***, la mejor universidad de Venezuela por ser responsable de mi formación universitaria.

Agradezco a la ***Asociación de Egresados y Amigos de la UCV***, por la ayuda económica brindada durante toda mi formación y por mejorar la calidad de mi educación.

Finalmente, agradezco a ***todas aquellas personas*** que de una u otra manera estuvieron presentes en mi vida durante este fructífero camino; conocidos, profes, panas, colegas universitarios y obreros de la Facultad de Ciencia… y demás personas que olvido mencionar pero que representan parte importante de mi logro.

Índice General

Índice de Figuras

Índice de Tablas

Introducción

El software de procesamiento gráficos 3D es el conjunto de aplicaciones que permiten la creación y manipulación de gráficos 3D por computadora. Estas aplicaciones son empleadas tanto para la creación de imágenes como en la animación por computadora. En la actualidad existen un número considerable de aplicaciones con propósitos muy diversos y de ámbito *standalone* y *online*.

Este crecimiento exponencial en la forma de percibir y crear contenido gráfico, ha generado cambios sustanciales en diversas áreas. En el último par de años, los navegadores han aumentado su rendimiento y son capaces de ofrecer una plataforma para albergar aplicaciones de procesamiento gráficos. La mayoría de los navegadores han adoptado la tecnología WebGL, que no sólo permite crear aplicaciones 2D y gráficos en el navegador, sino también crea aplicaciones 3D con un rendimiento óptimo, utilizando las capacidades de la unidad de procesamiento gráfico (GPU).

En la actualidad existen herramientas en la Web que mediante el uso de tecnología como WebGL y bibliotecas especializadas en el área de los gráficos 2D/3D, recrean un ambiente de desarrollo de aplicaciones gráficas Web. Asimismo, suelen incluirse avanzados procesadores de programas especiales para el procesamiento gráfico (*Shaders*), que posterior a su compilación son ejecutados de manera independiente por la GPU. Sin embargo, y basado en la investigación realizada, no existe un criterio unificado de implementación en los editores de *shaders* analizados, y en algunas aplicaciones en concreto no se percibe la incorporación de todos los elementos mínimos necesarios para llevar a cabo el proceso de desarrollo de *shaders*.

Este trabajo tiene como objetivo desarrollar un sistema integrado que permita el desarrollo de programas de sombreado para el lenguaje de OpenGL, haciendo uso de tecnología Web de vanguardia, y de arquitecturas pensadas para ofrecer un óptimo rendimiento. La finalidad de la solución (nombrada "*Shader Tool*"), es proveer una herramienta Web que incorpore y enlace todos los elementos involucrados en el desarrollo de programas GLSL, además de proporcionar accesos a la información generada en torno a los *shaders*.

Con el fin de lograr los objetivos establecidos, se ha estructurado el trabajo en seis capítulos que explicarán los diferentes aspectos que se tomarán en cuenta para la creación del sistema. A continuación, se ofrece un breve resumen del contenido de cada uno de estos capítulos:

En el primer capítulo se presenta la propuesta del Trabajo Especial de Grado donde se plantea la problemática existente en torno a las limitaciones existentes en los editores de *shaders* Web disponibles, se muestran en detalle los objetivos de la investigación, el alcance y la metodología que se seguirá para el desarrollo del sistema propuesto.

En el segundo capítulo se describen los fundamentos básicos relacionados a la tecnología de despliegue de gráficos por computadora en el ámbito Web. Se muestra el análisis de un grupo de editores de *shaders* actuales seleccionados, y se evalúan sus aspectos principales.

En el capítulo tres se detalla la arquitectura utilizada cliente-servidor y los actores que intervienen en el proceso de comunicación. Se explica la utilidad y conceptualización de una API REST que provea los servicios requeridos por la aplicación, y finalmente, de muestran los módulos funcionales de la aplicación del lado servidor y lado cliente.

En el cuarto y quinto capítulo se muestra el diseño general del sistema del lado servidor y cliente respectivamente. Se describe la aplicación, todas las bibliotecas dinámicas desarrolladas y la API REST implementada. Se abarcan los detalles de implementación de las funcionalidades y módulos más importantes.

El sexto capítulo muestra el análisis de los resultados obtenidos tras realizar las pruebas cualitativas del sistema, obtenidas mediante encuestas realizadas a un sub conjunto de la comunidad estudiantil de la Facultad de Ciencia.

Para finalizar, se presentan las conclusiones obtenidas durante la investigación y desarrollo, tomando en cuenta los resultados de las pruebas, y se sugiere posibles trabajos a futuro para la continuación de la investigación y desarrollo en el área.

Capítulo 1 Propuesta del Trabajo Especial de Grado

Se presenta a continuación la propuesta del Trabajo Especial de Grado, donde se definen las limitaciones existentes en la actualidad con respecto a la creación de *shaders* para el lenguaje de sombreado de OpenGL (GLSL). Se plantea el desarrollo de un sistema como objetivo general. Dicha aplicación será capaz de proveer las herramientas necesarias para llevar a cabo la implementación y pruebas de los *shaders* en un entorno Web. Adicionalmente, se especifica la metodología que se seguirá para el desarrollo del sistema propuesto.

1.1 Definición del problema

El desarrollo Web es parte importante de la evolución tecnológica actual y está presente en cada área de investigación posible. Herramientas como HTML, JS, CSS y WebGL ofrecen un amplio alcance en el desarrollo de una aplicación, y permiten construir aplicaciones con un alto impacto visual y funcional. Particularmente es de provecho evaluar la manera en que esta tecnología puede apoyar a la visualización y manejo de efectos, y en qué proporción puede mejorar los procesos y experiencia del usuario final.

Basado en el impulso brindado por la evolución de la tecnología Web en la última década, surge a principios de los años 90 los gráficos tridimensionales en la Web y las primeras iniciativas para promover el desarrollo de diversas herramientas en esta área. El auge del despliegue 3D en la Web hasta la fecha se ha fundamenta en la necesidad de contar con un ambiente de operación común para todos los usuarios y en la necesidad de tener una plataforma de desarrollo estándar para los desarrolladores de aplicaciones.

El despliegue 3D ha permitido el crecimiento sostenido de aplicaciones que ofrezcan de una u otra forma mayor facilidad para la manipulación de información gráfica. Dentro de este ámbito (aplicaciones gráficas en la Web), se encuentran las aplicaciones que brindan soporte a la implementación y prueba de programas para el lenguaje de sombreado de OpenGL (*Shader*) en las áreas de ingeniería, educación, arquitectura, finanzas, ventas y marketing, juegos y entretenimiento.

GLSL es el acrónimo de *OpenGL Shading Language* (Lenguaje de Sombreado de OpenGL), también conocido como GLslang, una tecnología parte del API [1]estándar OpenGL, que permite especificar segmentos de programas gráficos que serán ejecutados sobre la GPU. GLSL es un lenguaje de sombreado de alto nivel basado en el lenguaje de programación C en el cual se escriben unidades de códigos denominadas *Shaders*.

[1] Interfaz de programación de aplicaciones, la cual representa la capacidad de comunicación entre componentes de software

La tecnología *shaders* es cualquier unidad escrita en un lenguaje de sombreado que se puede compilar independientemente. Es una tecnología reciente y que ha experimentado una gran evolución destinada a proporcionar al programador una interacción con la unidad de procesamiento gráfico. Los *shaders* son utilizados para realizar transformaciones y crear efectos especiales, aplicados a gráficos generados por computadoras.

La manipulación de estos programas es una necesidad en muchas áreas de desarrollo e investigación, por su alto impacto y utilidad. En la actualidad existen soluciones Web implementadas por particulares, grupos o consorcios que de una u otra forma presentan mecanismos que permiten la creación de programas GLSL con objetivos muy diversos. Estas aplicaciones permiten crear *shaders* y manipular gran parte de las características asociadas a este, sin embargo, por la naturaleza del lenguaje y el alcance de dichos programas dentro de una escena 3D, el proceso de desarrollo se ve disminuido por la ausencia de una integración de criterios y herramientas.

En el proceso de desarrollo de efectos 3D pueden intervenir diferentes aspectos externos, que al conectarlos al programa desarrollado se produce la visualización de la escena. El manejo de mapas de bits y modelos son dos de estos aspectos y su soporte es limitado entre las diversas soluciones. La elección de la herramienta a utilizar para la implementación de programas GLSL en la Web se ve condicionada por la posibilidad de contar con el mayor número de características necesarias para el desarrollo.

De acuerdo a los requerimientos necesarios al momento de implementar un programa GLSL en la Web, el desarrollo se ve limitado al utilizar las herramientas existentes. El manejo de variables, malla de triángulos, texturas, datos de sesión, entre otros, es mínimo o está ausente. Estos fallos de funcionalidad y versatilidad ocasionan el uso de más de una herramienta de desarrollo o en su efecto la realización incompleta del programa. Así mismo, la posibilidad de almacenar bibliotecas de recursos personalizadas que esté disponible en el proceso de implementación del programa no es una opción actualmente.

Así, el desarrollo consistió en analizar, diseñar e implementar una solución que provee a los desarrolladores de una aplicación para la implementación de programas GLSL. El sistema es capaz de almacenar y procesar el código escrito en el lenguaje GLSL, emplea el despliegue de modelos 3D para la representación visual, obtenida tras la compilación exitosa del *shader*. Adicionalmente, le permite al usuario crear bibliotecas propias de recursos, asociarlos a sus programas, sincronizar su información de manera automática y visualizar de manera local la copia del efecto.

1.2 Objetivos de la investigación

1.2.1 Objetivo General

Desarrollar una solución integrada para el lenguaje de sombreado de OpenGL, utilizando tecnologías y herramientas Web.

1.2.2 Objetivos Específicos

- Analizar el proceso de desarrollo de los programas GLSL en la Web para determinar todos los elementos que intervienen y las limitaciones de las implementaciones actuales.
- Elaborar el diseño de las interfaces gráficas del usuario y definir toda la interacción del usuario con el sistema.
- Configurar y adaptar un editor de texto flexible adaptado a la sintaxis del lenguaje GLSL.
- Construir un contexto de despliegue para la visualización de modelos 3D y la interpretación de los programas GLSL.
- Modelar en 3D y cargar al sistema objetos estándar para utilizados en los programas de *shaders*.
- Desarrollar un sistema de inicio de sesión para asociar y resguardar los datos de los usuarios.
- Salvar de forma automática el contenido de los programas asociados a una sesión de usuario en particular.
- Implementar módulos de manipulación de texturas, modelos y *shaders*.
- Realizar las pruebas de la solución llamada *Shader Tool*.

1.3 Descripción de la investigación

1.3.1 Justificación

En la actualidad las aplicaciones disponibles y las herramientas que estas ofrecen para el desarrollo de programas de sombreado de OpenGL son insuficientes o en algunos casos inexistentes o poco prácticas e intuitivas. La implementación de *shaders* en estas plataformas resulta incompletas por no contar con todas las características necesarias y son propensas a procesos manuales repetitivos y pérdida de códigos.

En este trabajo se propone desarrollar una solución que permita contar con una plataforma Web de desarrollo de *shader*. Adicionalmente, la solución proporcionará un uso adecuado del espacio disponible de la ventana del navegador, un sistema de modales para mejorar la disposición de la información, una navegación simple e intuitiva y una interfaz gráfica de usuario que facilite tanto el aprendizaje de la herramienta como el uso de la misma.

El desarrollo de la solución Web generará beneficios expresados en la optimización del proceso de desarrollo de *shaders* y en la manipulación de recursos necesarios para su implementación, minimizando el tiempo requerido de desarrollo y la adaptación del programador al usar los diferentes módulos disponibles. Por último, la investigación pondrá de manifiesto los conocimientos adquiridos durante la carrera y permitirá sentar las bases para otros estudios que surjan partiendo de la problemática aquí especificada.

1.3.2 Delimitación y alcance de la investigación

La investigación se enfocará en el desarrollo de una solución computacional que permita la creación, manipulación y compilación de programas GLSL, así como su aplicación en modelos 3D en un entorno Web. Dicha solución dispondrá de una interfaz gráfica de usuario donde se muestra el código perteneciente al *shader* actual y la representación de este luego de la compilación (aplicado al modelo seleccionado). Es importante señalar que la solución permitirá la personalización del espacio de trabajo, el modelo y textura a utilizar en el programa GLSL, y la visualización de posibles errores presentes en el código del programa actual. El objetivo de dichas características es que el usuario tenga acceso a los diversos elementos que pueden intervenir en la implementación del efecto.

La interactividad del sistema permitirá a los usuarios manipular los elementos 3D desplegados a través de traslaciones y rotaciones haciendo uso de dispositivos de entrada estándar (teclado y/o ratón). Adicionalmente, los usuarios podrán cambiar las características del despliegue y las propiedades de la cámara de la escena 3D.

Aunado a esto, la solución contará con una serie de paneles y modales que facilitarán el acceso y utilización de los recursos disponibles, así como también la capacidad de cargar y desplegar modelos 3D e imágenes que posteriormente podrán ser asociadas a un programa GLSL, descargar una versión simplificada del efecto y la personalización de datos de perfil.

El sistema será diseñado para ser utilizado en ambiente Web en navegadores que brinden soporte nativo a WebGL y hará uso de la API de OpenGL 2.0 para el despliegue de gráficos 2D y 3D.

1.4 Metodología

- Se realizarán análisis de las soluciones actuales que están disponibles en la Web para la creación de programas GLSL para estudiar las limitaciones que estas presentan y definir las variables y requerimientos de la solución.
- Recopilación de los efectos estándar soportados en ambientes Web para la validación del sistema.
- Haciendo uso de un software se diseñará en 3D los distintos modelos que estarán disponible como parte de la biblioteca pública de la solución. El formato de importación empleado será JSON.
- Se utilizará el paradigma de Programación Orientada a Objetos (POO) con las adaptaciones pertinentes al lenguaje JavaScript. De esta forma todas las funcionalidades del sistema se construirán de manera modular, con una mayor abstracción y facilitará el mantenimiento del código fuente.
- Se utilizará la plataforma Web y los lenguajes de programación Javascript para el desarrollo de las funcionalidades del lado del cliente y PHP5 para brinda una API basado en REST que de soporte a la solución del lado del servidor.

- El enmarcado y estilo de la solución se hará utilizando HTML 5 y CSS3.
- Se estudiarán las distintas bibliotecas que existen para el despliegue de contenido 3D en la Web basados en WebGL y se seleccionará la que más se adecué a la solución.

Capítulo 2 Tecnología para el despliegue en la Web

El despliegue de gráficos 3D ha tomado un gran impulso en los últimos años, especialmente en el ámbito de los videojuegos. Debido a esto se han desarrollado APIs especializadas (interfaces de programación de aplicaciones) con el objetivo de facilitar los procesos en todas las etapas de generación de gráficos por ordenador. Estas APIs también han demostrado ser primordiales para los fabricantes de componentes gráficos de computadoras, ya que proporcionan un medio de acceso al *hardware* de una manera abstracta.

Desde hace aproximadamente dos décadas esta tendencia fue aumentando y se unió a la tecnología Web existente. Esto dio cabida a nuevas formas de producir gráficos 2D/3D y nuevas competencias y demandas en el ámbito de la computación gráfica.

La tecnología de despliegue Web fue tomada en consideración de manera formal y pública por primera vez en el año de 1994 cuando se crea VRML (*The Virtual Reality Modeling Language*), el cual es un formato de archivo que describe los objetos interactivos en un mundo 3D. VRML es diseñado para ser utilizado en Internet, Intranet y en sistemas que se basan en arquitecturas cliente-servidor, y para ser considerado como un formato de intercambio universal que integre gráficos 3D y multimedia. Este formato de archivo puede ser utilizado en aplicaciones varias y en áreas como la ingeniería, la visualización científica, en presentaciones multimedia, entretenimiento y educación, en páginas Web y en la creación de mundos virtuales sociales.

VRML hizo pública su primera especificación en el año 1995 (VRML 1.0). Dos años después pública su segunda especificación VRML 97 y da paso a X3D (*Extensible 3D*) como su sucesor, el cual fue creado por el consorcio Web3D (http://www.web3d.org). Éste consorcio fue formado para promover la creación de estándares abiertos para las especificación Web en la producción de gráficos 3D, y para acelerar la demanda mundial de productos basados en estos estándares.

En el nuevo estándar habilitado para Web presentado en X3D se integran gráficos 2D/3D, animaciones, interacción con el usuario, navegación, *networking*, objetos definidos por el usuario, *scripting*, simulaciones físicas, entre otras características que abren paso a una nueva posibilidad de desarrollo en aplicaciones gráficas en ámbito Web.

En la actualidad el esfuerzo gráfico tridimensional en la Web se ha centrado en *Word Wide Web Consortium (*http://www.w3.org/). W3C está constituido por una comunidad internacional que desarrolla estándares Web. El consorcio tiene como objetivo a largo plazo hacer las creaciones de escenas basadas en X3D mucho más naturales, de manera nativa y apoyadas por autores de la última versión de HTML (*HyperText Markup Language*).

Del mismo modo, Xj3D es un proyecto impulsado también por W3C y está enfocado en crear una suite de herramientas para VRML y X3D completamente desarrollada en Java. Esta suite puede ser usada para importar contenido VRML en aplicaciones personalizadas, entre otras cosas. La iniciativa para este proyecto fue crear un cargador de archivos para la API de Java3D con una versión alternativa de código de la empresa Sun Microsystems para el consorcio Web3D.

Por otra parte, desde el año 2004 existe otra propuesta a ser considerada. La propuesta de la empresa Sun Microsystems llamada Java 3D. Esta es una API 3D basado en una estructura de datos que permite la implementación de aplicaciones gráficas de edición vectorial para la plataforma de Java. Comparado con otras soluciones, Java 3D no es solo un *wrapper*[2] sobre una API gráfica, contiene una interfaz que encapsula la programación gráfica usando un enfoque orientado a objetos (*Object oriented programming*). Java 3D se integra y comunica con los navegadores a través de un plug-in (*Java plug-in technology*). Esto permite que las aplicaciones sean portables y aspectos de seguridad y eficiencia no sean labor de los navegadores.

Otras tendencias de desarrollo para la incorporación de gráficos 3D en la Web se enfocan en el uso de lenguaje como JavaScript (JS). Este lenguaje fue incorporado por primera vez en la versión 2002 del navegador Netscape en el año de 1995 y en la actualidad es el lenguaje de programación por excelencia del lado del cliente (*client side*). Próximo a la fecha de su públicación los desarrolladores evitaban el uso del lenguaje y gracias a la llegada de Ajax (*Asynchronous JavaScript And XML*), JS fue posicionándose entre los lenguajes interpretados más utilizados en la Web. Esto, resulto en un incremento sostenido en el desarrollo de *frameworks* y bibliotecas de ámbito general, mejorando así las prácticas de programación con el lenguaje.

Basados en la popularidad y potencial de JavaScript varios grupos de desarrollos coinciden que es el lenguaje Web que cuenta con la plataforma más sólida para ser considerado al crear estándares y APIs que permitan el desarrollo de aplicaciones gráficas en los navegadores modernos. Entre esos grupo se encuentra *Khronos WebGL Working* que desde el año de 2011 desarrolla la biblioteca WebGL (*Web Graphics Library*).

WebGL es una API escrita totalmente en el lenguaje JavaScript para el despliegue interactivo de gráficos 2D/3D sobre cualquier navegador compatible, y sin el uso de plug-ins. WebGL está integrado completamente dentro de todos los estándares Web para los navegadores, permitiendo el uso de aceleración a través de la GPU para cálculos físicos, procesamiento de imagen y efectos. WebGL es la solución para el despliegue de gráficos por computadora más aceptada en la actualidad por estar fundamentada en OpenGL, y por contar con la capacidad de estar presente en los navegadores de forma nativa. Basados en la tecnología que ofrece WebGL, han surgido innumerables bibliotecas de desarrolladores independientes que ofrecen una capa de abstracción para los programadores interesados en los gráficos 2D/3D.

[2] Programa o *script* que establece las bases y hace posible el funcionamiento de otro programa más importante. También conocido como programa o *script* de delegación.

Desde sus inicios WebGL ha promovido el desarrollo de bibliotecas y ha captado la atención de múltiples desarrolladores. Gracias a la popularidad del lenguaje JavaScript el número de herramientas existentes crece rápidamente. Al momento de construir una aplicación 2D/3D en la Web se cuenta con APIs y recursos para cálculos físicos, despliegue de contenido vectorial, canvas 2D/3D, simulaciones, y para resolver cualquier problema que pudiera presentarse en el desarrollo. Todo esto directamente desde el navegador y sin requerir instalaciones de aplicaciones o plug-ins.

Una de las herramientas por excelencia en el área de gráficos vectoriales es VML (*Vector Markup Language*) y SVG[3] (*Scalable Vector Graphics*). Ambos lenguajes se apoyan en nodos especiales DOM[4] (*Document Object Model*) que permiten representar formas como círculos, líneas y polígonos. Al igual que cualquier otro elemento HTML permite asociar estilos, ser posicionados a demanda e interacciones con el uso nativo de JavaScript.

En contraparte, la tendencia actual muestra un especial interés en el uso del elemento canvas propuesto en HTML5 para el manejo de contenido vectorial. Este nuevo elemento incorporado en la última versión de HTML permite la generación de composiciones de imágenes, gráficos dinámicamente, gráficos estáticos y animaciones por medio del *scripting* (ver **Figura 1**).

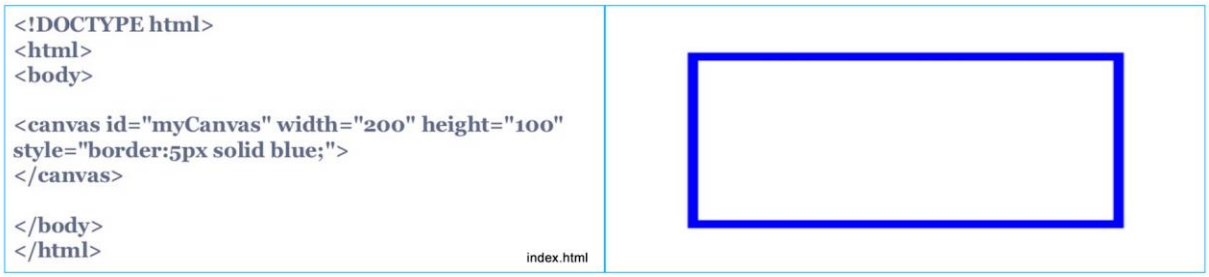


Figura 1 - Ejemplo Simple en un elemento Canvas HTML.

Entre las bibliotecas más populares que dan soporte y ofrecen mayor comodidad y accesibilidad al nuevo elemento HTML se encuentran bibliotecas como KineticJS (para mejorar el rendimiento de aplicaciones Web), EaselJS (simplifica el manejo del elemento), Paper.js (biblioteca de generación de gráficos vectoriales), Fabric.js y Processing.js (para el procesamiento de datos en aplicaciones Web). Todas desarrolladas sobre JavaScript.

Actualmente, existe otra biblioteca llamada Three.js y desde su aparición en el año 2010 ha mantenido un crecimiento constante. Probablemente esta tendencia de crecimiento se deba a que está basada en dos de las herramientas más estables y estandarizadas en el ámbito Web, WebGL y JavaScript.

[3] Especificación que describir gráficos vectoriales bidimensionales, tanto estáticos como animados.
[4] Interfaz de programación de aplicaciones (API) que proporciona un conjunto estándar de objetos para representar documentos HTML y XML.

Three.js detalla una amplia API que permite la creación de animaciones 3D aceleradas por GPU que utilizan el lenguaje JavaScript como parte de un sitio Web sin depender de plug-ins. La biblioteca permite el uso de canvas, transformaciones 3D CSS[5] (Cascading Style Sheets), efectos varios, imágenes en formato anáglifo, definición de escenas, cámaras perspectivas y paralelas, controles, luces, materiales, empleo de *shaders*, sonidos, mallas de polígonos, y otras utilidad. Con más de 30 características relativas a la construcción de aplicaciones y juegos 3D, Three.js puede considerarse como un motor gráfico 3D basado en tecnología Web.

Toda esta evolución gráfica es posible gracias a que los motores de JavaScript actuales son lo suficientemente eficientes para ejecutar juegos 3D y manipular videos en tiempo real. Adicionalmente, la composición acelerada por *hardware* también se está implementando en muchos navegadores, lo que implica que incluso las transformaciones y transiciones con CSS serán rápidamente ejecutadas en el equipo local.

Las bibliotecas de JavaScript, los *frameworks* CSS, el contenido multimedia y demás avance en el área del software Web van de forma síncrona con el lenguaje de marcado HTML5. Este lenguaje es primordial en el desarrollo Web, y hoy en día es la base para la construcción de aplicaciones Web de diversa naturaleza.

2.1 HTML5

A principio del año 2000 se incluyó en los navegadores mayor interacción, permitiendo que las partes de una página pudiesen cambiar dinámicamente a través de la técnica AJAX. Sin embargo, las formas en que las páginas se podían actualizar se vieron limitados por las características gráficas de HTML y CSS. Hoy en día esa realidad ha cambiado y el navegador Web se ha convertido en una plataforma capaz de ejecutar aplicaciones sofisticadas de forma nativa con un alto rendimiento. HTML5 representa una revisión masiva al estándar HTML, incluyendo la limpieza de sintaxis, las nuevas características del lenguaje JavaScript y APIs, las capacidades móviles y soporte multimedia avanzado. La plataforma HTML5 ofrece un conjunto de tecnologías de gráficos avanzados tales como:

- **WebGL**: Aceleración 3D por *hardware* implementado en JavaScript. Basado en la API de OpenGL. El estándar es compatible con la mayoría de los navegadores Web.

- **CSS3**: Permite aplicar transformaciones 3D, transiciones y filtros personalizados al contenido de un sitio Web. CSS ha evolucionado en los últimos años para incluir la representación 3D y funciones de animación, accesibles a través del lenguaje de hojas de estilo.

- **El elemento Canvas y su API de dibujo 2D**: Universalmente apoyado en los navegadores, esta API desarrollada bajo JavaScript, permite a los desarrolladores dibujar gráficos arbitrarios en

[5] Lenguaje de hojas de estilo utilizado para describir el aspecto y el formato de un documento escrito en un lenguaje de marcas

la superficie de un elemento DOM. Aunque el canvas utiliza una API 2D, con la ayuda de bibliotecas de JavaScript adicionales se puede lograr efectos que simulen la perspectiva 3D.

Cada una de estas herramientas tiene un papel que desempeñar en la entrega de experiencias interactivas en 3D y visualmente atractivas.

HTML5 incorpora grandes cambios para el manejo de gráficos en la Web, sin embargo esto no sería tan notorio sin la presencia de otras mejoras esenciales del navegador. En particular, varios de los avances han marcado un camino en el desarrollo de aplicaciones ricas en elementos HTML5:

- **Máquina virtual (VM) JavaScript de alto rendimiento**: WebGL y Canvas 2D son APIs implementadas en JavaScript. Hace algunos años, el rendimiento de la máquina virtual habría hecho del desarrollo 3D un proyecto no viable. Actualmente, las máquinas virtuales actuales ofrecen el rendimiento necesario.

- **Composición acelerada**: El navegador es responsable de la combinación o composición de los diversos elementos de una página Web de forma rápida y sin fallos visual u otros efectos adversos. Como el contenido se ha vuelto más dinámico, los desarrolladores han incluido grandes mejoras en sus navegadores, incluyendo el uso del *pipeline* gráfico por *hardware* para el despliegue de todos los elementos visuales, tanto en 2D como en 3D.

- **Soporte de Animación**: La función *requestAnimationFrame* se introdujo como una alternativa al uso de *setInterval* o *setTimeout* para conducir animaciones, en gran medida para mejorar el rendimiento y la eliminación de artefactos visuales.

Los navegadores con soporte para HTML5 también incluyen características de programación multi-hilo (*Web Workers*[6]), la creación de redes dúplex TCP/IP completo (*WebSockets*), almacenamiento de datos local, entre otras utilidades a disposición de los desarrolladores. Estas características, en conjunto con WebGL, CSS3 3D y el elemento canvas, representan una nueva y amplia plataforma para la entrega de aplicaciones visuales conectadas en cualquier computadora o dispositivo.

2.2 WebGL

El futuro del desarrollo Web tiene diferentes caminos, esta se determinada en función de las tecnologías que se utilice. El objetivo siempre es ofrecer contenidos varios y accesibles a los usuarios en formatos visualmente más llamativos.

La creación de aplicaciones 3D en la Web pertenece a un área de la computación bien conocida y ampliamente difundida. En esta rama se encuentras paquetes de programación como por ejemplos

[6] Código Javascript que se ejecuta en background de forma independencia de otras secuencias de comando.

Adobe Flash, Unity3D o Microsoft Silverlight, los cuales representan opciones de desarrollos atractivas y funcionales. Su principal desventaja es la necesidad de instalar *plug-ins* para habilitar sus funciones en el navegador.

Otro sistema de software común que proporciona aplicaciones 3D en el navegador Web son los Applets[7] de Java. Estos no requieren la instalación explícita de algún *plug-in*, sin embargo requieren tener la máquina virtual de Java instalada.

La tecnología WebGL evita esos inconvenientes. Un usuario puede navegar en un entorno 3D directamente en un navegador Web compatible con WebGL (ver **Figura 2**). Esta comodidad implica complicaciones adicionales para el desarrollador de una aplicación gráfica, ya que la biblioteca sólo proporciona una interfaz de programación de aplicaciones rudimentarias.

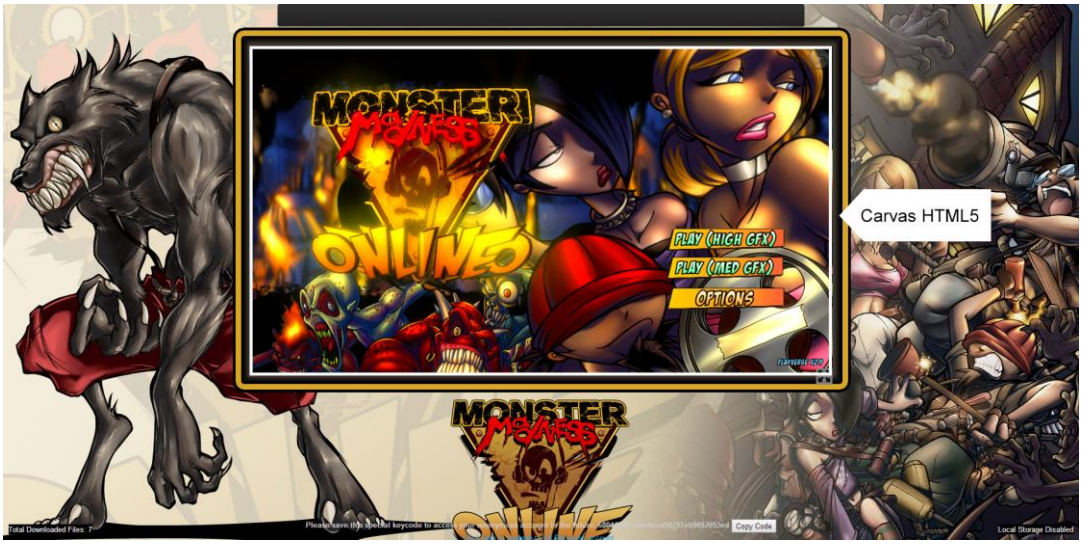


Figura 2 – Ejemplo de un entorno 3D interactivo WebGL.

WebGL es una especificación estándar que está siendo desarrollada para desplegar gráficos en 3D en navegadores Web. Permite activar gráficos en 3D acelerados por *hardware* en páginas Web, sin la necesidad de plug-ins en cualquier plataforma que soporte OpenGL 2.0 u OpenGL ES 2.0. Técnicamente es un enlace (*binding*) para JavaScript que permite el uso de la implementación nativa de OpenGL ES 2.0 en los navegadores Web.

El soporte que pueda tener un cliente de WebGL está definido principalmente por el tipo de navegador que se utilice, y por el rendimiento del *hardware* local del usuario. Desde sus primeras versiones, los desarrolladores de navegadores y principalmente los más conocidos y usados (Mozilla Firefox, Google Chrome, Opera y Safari) se han esforzado en ir al ritmo del grupo de desarrollo WebGL.

[7] Un applet es un componente de una aplicación que se ejecuta en el contexto de otro programa, por ejemplo en un navegador Web.

2.3 Definición técnica

WebGL es desarrollado y mantenido por el grupo Khronos, el organismo de normalización que también mantiene OpenGL, COLLADA y otras especificaciones. Esta es la descripción oficial de WebGL, desde el sitio Web Khronos:

"WebGL es una API multiplataforma bajo un estándar libre de bajo nivel para el manejo de gráficos 3D basados en OpenGL ES 2.0. Se expone utilizando el elemento canvas de HTML5. Utiliza el lenguaje de sombreado de OpenGL, GLSL ES, y se puede combinar sin problemas con otro tipo de contenido Web. Es ideal para aplicaciones Web dinámicas en 3D basadas en el lenguaje de programación JavaScript. Se integrará plenamente en los principales navegadores Web."

Esta definición comprende varias ideas centrales:

- WebGL es una API. Está se accede exclusivamente a través de un conjunto de interfaces de programación de JavaScript. Representación 3D en WebGL es análogo al dibujo 2D utilizando el elemento canvas, en donde todas las interacciones se manipulan a través de llamadas a la API de JavaScript. Ofrece conexión a WebGL usando el elemento canvas existentes y la obtención de un contexto especial de dibujo específico para WebGL.

- WebGL está basado en OpenGL ES 2.0 y es una adaptación del estándar OpenGL. El termino "ES" es sinónimo de "sistemas integrados", lo que significa que se ha adaptado para su uso en pequeños dispositivos de cómputo, como teléfonos y tabletas. OpenGL ES es una interfaz de programación de aplicaciones que permite gráficos 3D en dispositivos móviles de diversos fabricantes y sistemas operativos. **Figura 3** se muestra la arquitectura de despliegue de gráficos generados por computador en el entorno Web.

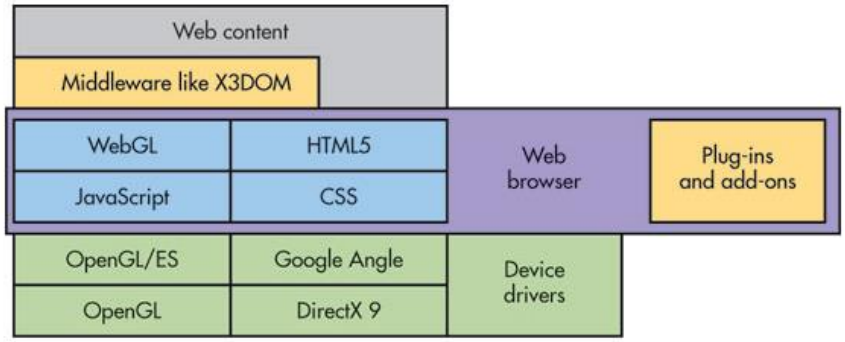


Figura 3 – Arquitectura Web de despliegue de gráficos generados por computador.

WebGL se combina con otro tipo de contenido Web. La capa de WebGL puede estar por encima o debajo de otros contenidos Web pertenecientes a una página dada. El elemento canvas 3D pueden ocupar la proporción necesaria.

- WebGL fue construido para desarrollar aplicaciones Web dinámicas. Se ha diseñado con el soporte Web en mente. La biblioteca ha sido adaptada con características específicas que se integran con los navegadores Web y con la capacidad de trabaja con el lenguaje JavaScript.

- WebGL es capaz de funcionar en cualquier sistema operativo y en dispositivos móviles y de escritorio.

2.4 Antecedentes

La Web ha sido siempre un medio visual, sin embargo, hasta hace poco fue de uso exclusivo de un grupo de aplicaciones específicas. La mayoría de desarrollos realizados eran sitios Web basados enteramente en CSS y JavaScript, sin embargo, con la incorporación de tecnologías como el elemento canvas, la especificación WebGL y las imágenes SVG el desarrollo Web ahora es más dinámico y adaptable. El uso combinado de toda esta tecnología puede dar como resultados aplicaciones, juegos o herramientas de alto impacto. La Web ofrece posibilidades de uso diversos y los editores de *shaders* es uno de ellos.

Hasta hace poco el editor de texto era un programa que permitía exclusivamente crear y modificar archivos digitales compuestos por texto sin formato, conocidos comúnmente como archivos de texto o texto plano. La forma de evaluar los editores de texto ha cambiado, y ya es posible en pocos segundos contar con un editor listo para su uso sin instalaciones o configuraciones iniciales, todo en la Web. Este tipo de herramientas evoluciono y de la mano de WebGL ofrecen editores especializados para la manipulación de *shaders*.

En WebGL se puede proporcionar 2 programas de *shaders* (*Vertex Shader* y *Fragment Shader*) que se invocan en el momento apropiado en el *pipeline* de OpenGL. El *shader* de vértices proporciona las coordenadas para cada vértice que se elaborará, y el *shader* de fragmento proporciona el color de cada píxel a ser dibujado.

Estos *shaders* están escritos en *OpenGL Shading Language* o GLSL. En WebGL pueden ser incluidos en una página Web de varias maneras: como texto codificado en un archivo fuente de JavaScript, como archivos independientes incluidos con el uso de la etiqueta <script>, o se recupera desde el servidor como texto sin formato. El código JavaScript se ejecuta en la página y luego se envía para su compilación utilizando la API de WebGL. Finalmente son ejecutados en la GPU del dispositivo.

Los desarrolladores de los editores de *shaders* que existen hasta la fecha se esfuerzan por ofrecer la combinación más adecuada entre una aplicación Web y un procesador de gráficos 3D. El nivel de innovación en cada implementación varía, sin embargo, características sociales y la creación de comunidades son las más usuales. De una u otra manera la influencia de la Web y la tendencia de compartir el contenido que se encuentra en la red definirá la forma en que se desarrollan aplicaciones gráficas.

A continuación se mostrara una revisión del estado del arte para los editores de *shaders* actuales disponibles en la Web.

2.4.1 Editor SHDR

SHDR es un visor, editor y validador de *shaders* en línea el cual fue desarrollado por Thibaut Despoulain (ver **Figura 4**). Esta soportado por WebGL, Three.js, Ace.js, RawDeflate.js y jQuery. SHDR combina parte de la tecnología en línea comúnmente aplicada solo a páginas o sistemas Web y la vertiente actual en herramientas gráficas en la nube.

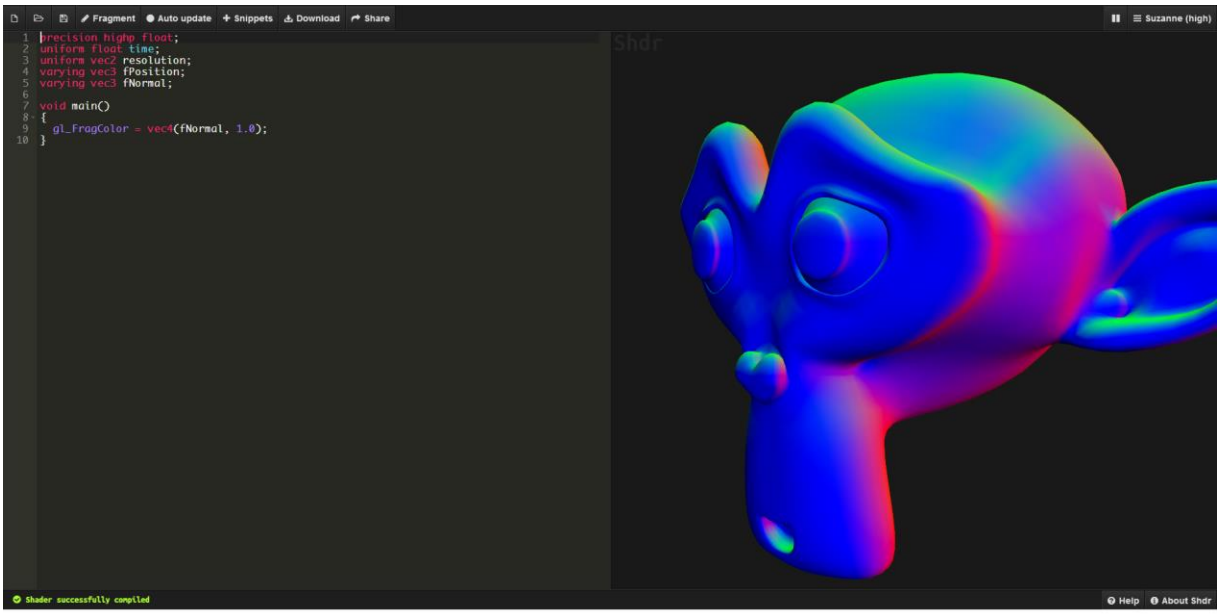


Figura 4 - Interfaz inicial del editor SHDR.

El editor de texto cuenta con indicadores visual (*code highlight*), sombreado de línea seleccionada, *snippets*[8], cambio de contexto entre *Shaders* y opciones comunes de manejo de archivo como descargar, abrir y guardar el trabajo actual. Por otra parte, el despliegue es aplicado en modelos 3D descargados al equipo local desde el servidor.

El ámbito de trabajo es completamente 3D y permite manipular libremente el *shader* de vértice y de fragmento con actualización automática al editar. El alcance de este rubro está limitado al ingenio que pueda darle el color del fragmento al modelo, características como uso de texturas, configuración de variables de entrada al shader, *geometry shader* (abreviado GS[9]), entre otras, no son soportadas.

Características Web de vanguardia también son aplicadas en el editor. En SHDR se exhibe el uso de CSS avanzado, AJAX como mecanismo de comunicación asíncrono, modales y paneles. Estas y

[8] Pequeña parte reusables de código fuente, código binario o texto.
[9] Modelo de programación de *shader* introducido con *Shader* Model 4.0 de DirectX 10.

otras características son heredadas de la evolución actual de las herramientas Web disponibles, y son aplicadas sin requerir software adicionales UI/UX[10] como es lo habitual en aplicaciones de escritorio.

2.4.2 GLSL Sandbox

Esta aplicación fue pensada principalmente para crear una comunidad de programadores y compartir y distribuir *shaders*, sin embargo, cuenta con herramientas de edición y despliegue de efectos que lo categorizan como un editor. La principal fortaleza de Sandbox es su amplia galería de efectos y su visor en pantalla completa (ver **Figura 5**).

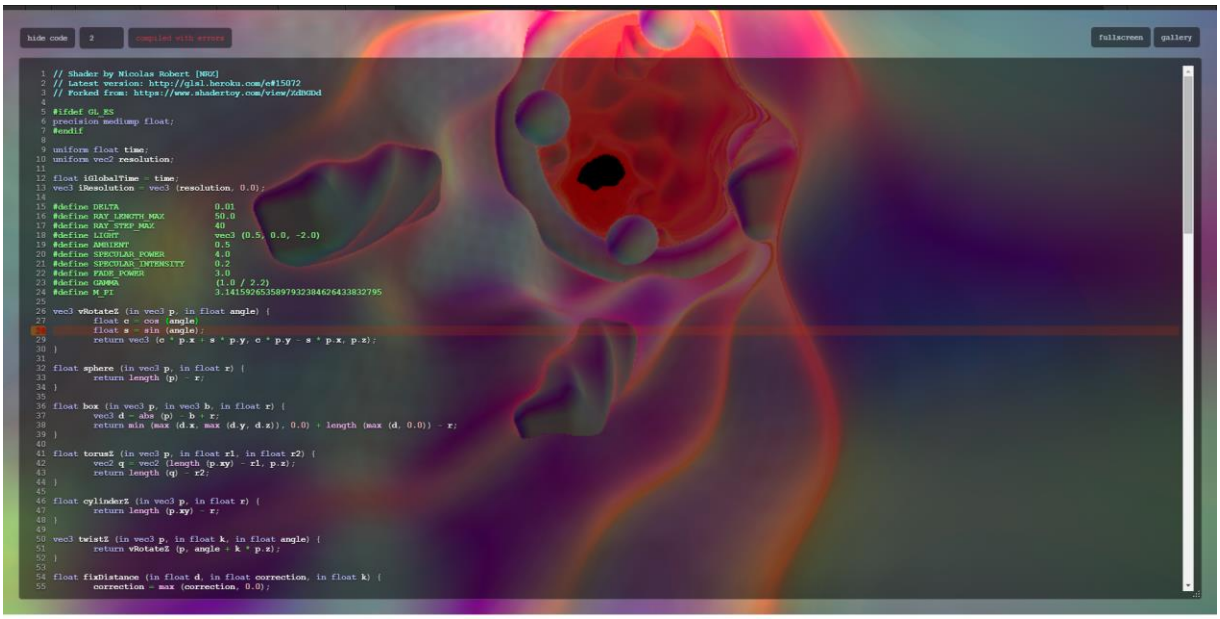


Figura 5 - Interfaz inicial del editor GLSL Sandbox.

GLSL Sandbox es limitado y poco práctico en la edición de los efectos, ya que no cuenta con un editor de código real. En contraste, ofrece la posibilidad de acceder a prácticamente cualquier sección o módulo involucrado en el proceso de despliegue. Esto permite que el visor de despliegue sea 2D o 3D, el uso de una malla de triángulos propia a través de *buffers* de despliegue, interacción con los dispositivos de entrada del equipo local, manipulación de las propiedades de la cámara, entre otras.

2.4.3 Shader Lab

Shader Lab (SL) es un editor de efectos experimental que nace de las limitaciones propias del desarrollo nativo y la característica multiplataforma deseable entre equipos Unix y Windows. SL es bastante limitado en cuanto a características de usabilidad como se muestra en la **Figura 6**. Las opciones de edición de código son prácticamente ninguna, ofreciendo 3 áreas de texto simples, sin indicadores visuales o alguna característica que potencie la experiencia de los usuarios.

[10] Diseño de experiencias de usuario.

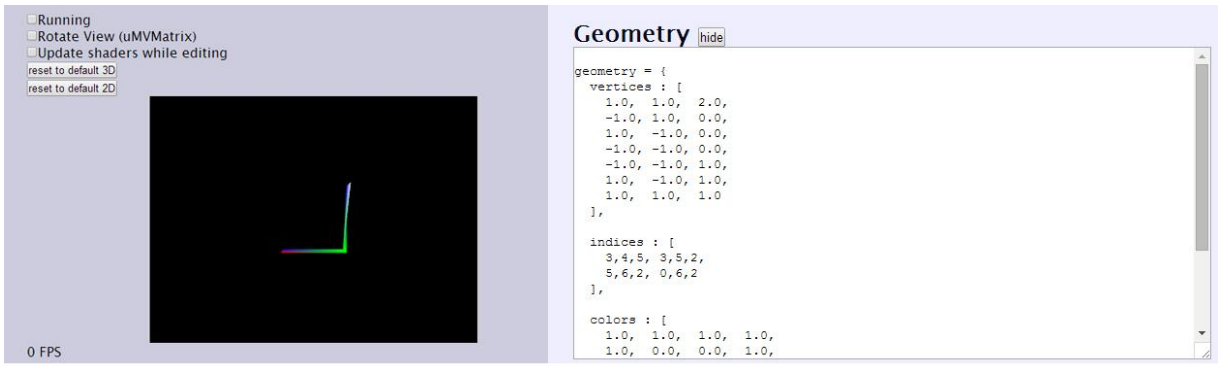


Figura 6 - Interfaz inicial del editor Shader Lab.

Aunque el editor se encuentra en desarrollo, son muchos los aspectos que están ausentes en la versión actual. Es necesario un adecuado manejo del texto que componen los *shaders*, mayor retroalimentación del estado del efecto, crear módulos que favorezcan a la interfaz de usuario, utilizar mayores características de HTML5 y CSS3 que mejoren el aspecto funcional y visual de la aplicación, recursos y/o efectos de muestra, entre otras opciones.

Entre sus características a destacar se pueden mencionar el reinicio del contexto de trabajo del elemento canvas, actualización automática del resultado de los *shaders* editados, despliegue de la velocidad de despliegue, uso de *geometry shaders* y aplicación de transformaciones básicas. SL utiliza WebGL de forma nativa para su implementación.

2.4.4 Kick.js

Kick es una biblioteca para navegadores Web modernos. Kick.js provee una API que permite crear una abstracción de WebGL y así construir aplicaciones o juegos en los navegadores de manera más sencilla. Basados en esta API sus desarrolladores diseñaron y desarrollaron un editor de efectos 3D lo suficientemente estable y completo basado en las características fundamentales que puedan ser necesarias.

El editor de *shader* presenta una interfaz de usuarios simples sin mayor innovación en cuanto a su aspecto visual. No obstante, en cuanto a funcionalidades es el editor más completo que en la actualidad se encuentra disponible en la Web de acuerdo a la investigación. Sus módulos novedosos son el uso de texturas múltiples, ajuste de variables (tipo *uniform*), panel de reporte de estado de efectos y configuraciones varias del contexto 3D. Adicionalmente permite cargar y almacenar efectos, crearlos desde su inicio o usar alguno de los ejemplos incluidos en la aplicación. Muchas de estas características se muestran en la **Figura 7**.

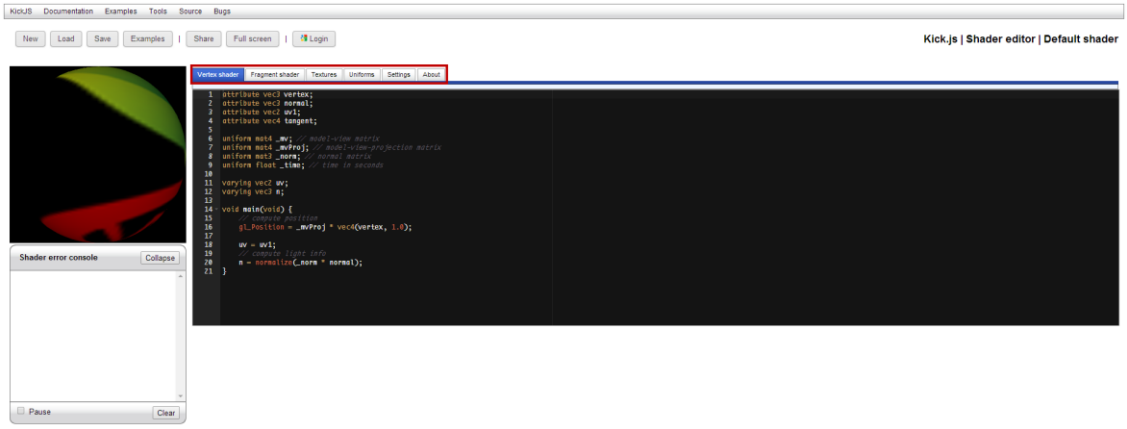


Figura 7 - Interfaz inicial del editor Kick.js.

Una característica especial es el inicio de sesión que permite ingresar empleando una cuenta de usuario de Google. De esta forma se puede almacenar y restaurar el trabajo de los usuarios fácilmente. Sin duda, esta característica es factible desde cualquier otra aplicación gráfica de escritorio, sin embargo, el hecho de contar con el entorno de desarrollo Web facilita considerablemente esta y otras tareas similares.

Por ser una API de desarrollo la documentación es muy completa y se cuenta con guías de instalación y uso de la biblioteca. De acuerdo a las características incluidas en el *release* actual se encuentran la gestión de teclado y mouse, uso de *skybox*, tipos de luces principales, texturas, *picking*, mapas de sombra, serialización y otras.

2.4.5 ShaderToy (ST)

ST fue la primera aplicación de su tipo que le permitió a los desarrolladores impulsar píxeles de código en la pantalla de un navegador Web usando WebGL.

La plataforma desarrollada incluye compilación en tiempo real del código del *shader*, una amplia comunidad de desarrolladores, discusiones abiertas por medio de comentarios, insumos para los *shader*: videos, webcam, audio, texturas, hora del día y la posición del cursor, visualización en pantalla completa, editor con resaltado de sintaxis, guardar, compartir y publicar los *shaders* (ver **Figura 8**). El alcance de la herramienta es amplio, es posible desarrollar efectos simples como animaciones de *sprites*[11] o composiciones 3D complejas tales como terrenos procedurales. El conocimiento técnico para ser capaz de usar la herramienta es alto.

[11] Mapas de bits que representa a algún objeto y su visualización en secuencia da la ilusión de movimiento.

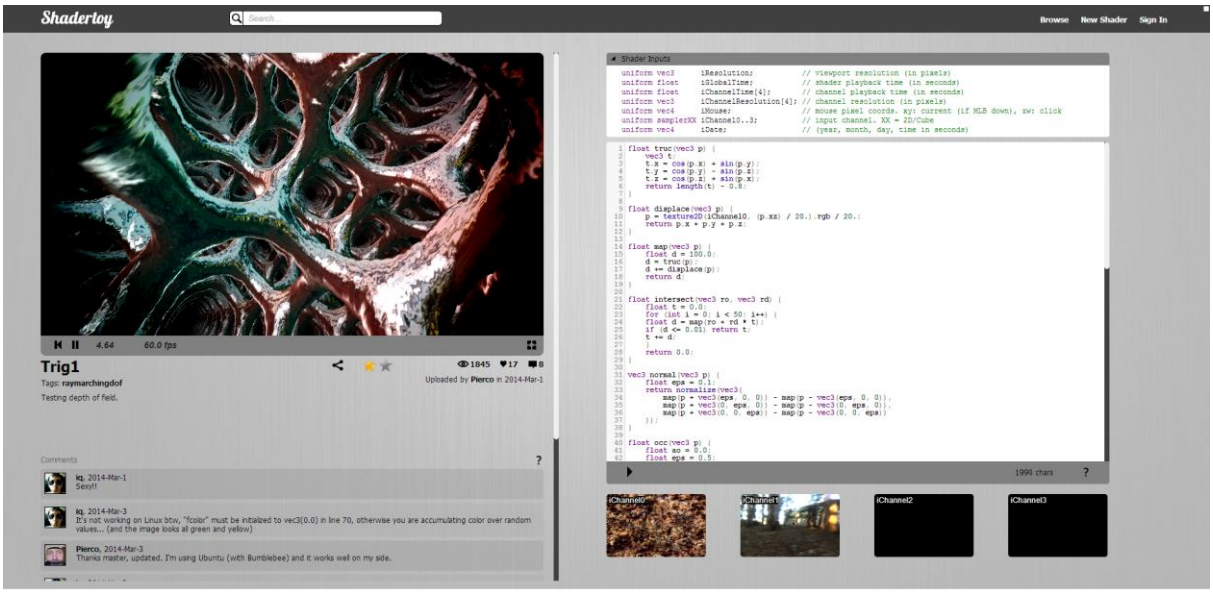


Figura 8 - Interfaz inicial del editor Shadertoy.

La apuesta definitiva de ShaderToy es el uso de las comunidades online. La posibilidad de contar con una amplia galería que se mantiene y gestiona por si sola es la mayor ventaja que la Web le ofrece. Compartir, difundir y publicar contenido con un solo clic transforma la aplicación en una plataforma de desarrollo y colaboración múltiple disponible en todo momento.

2.5 Estudio comparativo

Son varias las etapas y requerimientos de software y *hardware* que se necesitan para desarrollar un programa GLSL. Aunado a ello, al llevar a cabo este proceso empleando tecnología Web hay que tener en cuenta que existen aspectos adicionales a considerar. Seguridad en la red, velocidad de trasferencia, estilo gráfico, manejo de sesión y redes sociales son algunos de estos aspectos. Las aplicaciones analizadas y evaluadas anteriormente cuentan con un sub grupo de estas características.

A continuación se presenta la comparativa de dichas aplicaciones (ver **Tabla 1**), teniendo en cuenta los aspectos más relevantes al tratarse de un editor de *shader* en línea. El objetivo es evaluar de manera más explícita sus características y las posibles fallas o carencias que estas presentan.

2.5.1 Datos básicos

	SHDR	GLSL Sandbox	Shader Lab	Kick	Shader Toy
Web	shdr.bkcore.com	glslsandbox.com	codedstructure.net	tinyurl.com/kickjs-org	shadertoy.com
Ayuda	✔	✖	✖	✔	✔

Tabla 1 - Editores de *Shaders* en línea analizados.

2.5.2 Editor de texto

Las características que ofrezca el editor al momento de programar influyen en gran medida en la velocidad y fluidez en la que el proceso se lleva a cabo. En el ámbito *Standalone*[12] las opciones son amplias y existen editores con un potencial en opciones y funcionalidad alto. En lo que se refiere a la Web, aun los editores están en evolución, sin embargo, existen bibliotecas (principalmente escritas en JavaScript) que ofrecen posibilidades atractivas, útiles y de gran alcance. Aquí se evalúan varias de esas características aplicadas a los editores disponibles en cada aplicación analizada (ver **Tabla 2**).

	SHDR	GLSL Sandbox	Shader Lab	Kick	Shader Toy
Auto completado	✗	✗	✗	✗	✗
Validador de error	✓	✓	✓	✓	✓
# de línea	✓	✓	✗	✓	✓
Snippets	✓	✗	✗	✗	✗
Descargar código	✓	✗	✗	✗	✗
Indicador gráficos	✓	✓	✗	✓	✓
Editor de parámetros in	✗	✗	✗	✓	✗
Comodidad de edición	5/6	3/6	1/6	5/6	3/6
Auto guardado y despliegue	✓	✓	✓	✓	✓
Colapso se secciones	✓	✗	✗	✓	✓
Búsquedas	✗	✗	✗	✗	✗

Tabla 2 – Comparativa de aplicaciones analizadas (Editor de texto).

2.5.3 Canvas y utilidades

El código realizado para cada uno de los programas GLSL debe ser compilado y representado en un área apta para la manipulación de contenido 3D. En esta etapa es deseable contar con modelos que faciliten esta representación, y utilidades como el manejo de eventos y trasformaciones aplicadas a estos. Adicionalmente, se podría proporcionar información de rendimiento y estado del proceso. En la **Tabla 3** podrá visualizar la evaluación realizada en esta categoría.

[12] Programa informático que puede trabajar sin conexión a una red.

	SHDR	GLSL Sandbox	Shader Lab	Kick	Shader Toy
Indicador FPS	✗	✗	✓	✗	✓
Ejemplos / Modelos	4/6	5/6	1/6	4/6	*5/6*
Transformaciones básicas	✓	✗	✗	✗	✗
Detalles de despliegue	✗	✗	✗	✗	✗
Manejo de eventos	✗	✗	✗	✗	✓
tecnología de despliegue	threejs.js	glsl.js	WebGL nativo	kick.js	WebGL nativo

Tabla 3 - Comparativa de aplicaciones analizadas (Area de despliegue y otras).

2.5.4 Despliegue (*Render*)

Posteríos a la compilación y despliegue de los programas de *shaders*, las características y variables que ofrece el proceso son variadas. Es esta etapa se ven involucrado la aplicación de transformaciones 3D como rotaciones y traslaciones, opciones de cámaras, manejo de luces, entre otras. La flexibilidad que ofrezca el editor en este reglón define en gran medida el alcance de la herramienta (ver **Tabla 4**).

	SHDR	GLSL Sandbox	Shader Lab	Kick	Shader Toy
Contexto 2D/3D	3D	2D Efectos	3D	3D	2D – 3D
Uso de texturas	✗	✗	✗	✓	✓
Ajuste Modo de despliegue	✗	No aplica	✗	✓	✗
Ajuste Cara a desplegar	✗	No aplica	✗	✓	✗
Despliegue de normales/BB	✗	No aplica	✗	✗	✗
Ajuste cambiar Sombreado	✗	No aplica	✗	✗	✗
***Shaders* (G-F-V)**	F-V	F	F-V	F-V	F
Conf. de Cámara	✗	✗	✗	✗	✗
Manejo de Luces	✗	✗	✗	✓	✗
Formatos admitidos	Privado	No aplica	Manual Texto	Privado	Código
Biblioteca de recursos	✗	✗	✗	✓	✓
Galería de efectos	✗	✓	✗	✗	✓

Tabla 4 - Comparativa de aplicaciones analizadas (Proceso de Render).

2.5.5 Experiencia de usuario y utilidades

Las aplicaciones *Standalone* suelen tener una navegación estándar y conceptos de usabilidad pre establecidos de cierto modo por el lenguaje o IDE empleado para el desarrollo. En el entorno Web se involucran conceptos y variables adicionales que hace más difícil plantear el esquema y estructura de la aplicación. Conceptos como el diseño Web adaptable (*Responsive*) o el soporte multi-navegador (Cross Browser) presentan retos en el proceso de implementación y en la posibilidad de innovación en el área de despliegue de gráficos 3D. El soporte de estos aspectos es presentado en la **Tabla 5**.

	SHDR	GLSL Sandbox	Shader Lab	Kick	Shader Toy
Tipo de navegación	Ninguna	Jerárquica	Ninguna	Ninguna	Jerárquica
***Layout* flexible**	✔	✔	✘	✘	✔
Interacción con periféricos de entrada/Salida	✘	✘	✘	✘	✘
Retroalimentación Visual	✔	✘	✘	✘	✔
Zoom óptico	✘	✘	✘	✘	✘
Compartir	✘	✘	✘	✔	✔
Selector de archivo	✘	✘	✘	✘	✔
Controles	✔	✔	✘	✔	✔
Menú contextual	✘	✘	✘	✘	✘
Pantalla completa	✘	✔	✘	✔	✔
Manejo de sesiones	✘	✘	✘	✔	✔

Tabla 5 - Comparativa de aplicaciones analizadas (Experiencia de usuario y utilidades).

Capítulo 3 Sistema Integrado de desarrollo para el lenguaje de sombreado de OpenGL

Para cumplir con la propuesta realizada se analizó y seleccionaron las arquitecturas y patrones de diseño de software que se adaptaban mejor a los objetivos establecidos. Teniendo en cuenta que la solución será accedida de manera *online*, el planteamiento se enfocó exclusivamente en el ámbito Web. A nivel estructural la solución emplea el uso de la arquitectura de dos niveles cliente-servidor, REST y un sistema centralizado de datos.

Cada nivel de la solución se encuentra dividida en módulos que atienden una necesidad en particular. Del lado del servidor se atienden aspectos como el acceso a los datos, la representación de las entidades y la presentación y respuesta de los recursos solicitados. Por otra parte, en el lado del cliente se incluyen módulos para la optimización de la trasferencia de datos, diseño de componentes Web, enrutador, entre otros.

A continuación se muestra la descripción de las arquitecturas de desarrollo de software incorporadas en la solución, sus módulos funcionales y el modelo de comunicación implementado para el intercambio de información entre ambos niveles (cliente-servidor). De igual forma, se precisan conceptos importantes en torno a los patrones de diseño de software presentes en la solución.

3.1 Aplicación Web

Las aplicaciones Web usan la infraestructura de la Web (protocolos, lenguajes, entre otras) para su funcionamiento. Las componentes principales que la conforman son los navegadores y servidores Web. Esto permite tener acceso a la interfaz de usuario de manera global. Entre las ventajas que ofrece esta plataforma se encuentran su alto nivel de accesibilidad, soporte multiplataforma y facilidad en el mantenimiento.

Las aplicaciones Web son programas distribuidos que usan las tecnologías Web como su infraestructura. Usan un navegador Web como clientes, el protocolo HTTP para comunicarse entre clientes y servidores, y el lenguaje HTML para expresar el contenido transmitido entre clientes y servidores.

Debida a la naturaleza de la solución y el interés de aplicar métodos y herramientas actuales, la taxonomía Web utilizada en el desarrollo fue la orientada a servicio. En este tipo de solución se ofrecen servicios especializados, por lo cual se implementó una lógica de negocio acorde a cada uno de ellos.

3.2 Arquitectura cliente-servidor

La solución está diseñada en dos niveles los cuales corresponden uno al lado del cliente y el otro al lado del servidor. Este modelo ofrece características adaptables en cuanto a usabilidad, flexibilidad, interoperabilidad y escalabilidad en las comunicaciones. Su funcionamiento es sencillo: un equipo de cómputo cliente, requiere de un servicio de un equipo servidor, éste realiza las funciones para la cual fue programado para dar respuesta a la solicitud. Es importante destacar que ambos entes pueden residir en la misma máquina y aun así establecer la comunicación.

Desde el punto de vista funcional, se puede definir el modelo cliente-servidor como una arquitectura distribuida que permite a los usuarios finales obtener acceso a la información en forma transparente aún en entornos multiplataforma.

En este modelo, el cliente envía un mensaje solicitando un determinado servicio a un servidor (realiza una petición), y este envía uno o varios mensajes con la respuesta (Ver **Figura 9**).

Figura 9 - Esquema de comunicación Cliente-Servidor.

La mayoría del trabajo de lógica de negocio la realiza el proceso servidor y los procesos cliente sólo se ocupan de la interacción con el usuario (aunque esto puede variar). En otras palabras la arquitectura cliente-servidor es una extensión de programación modular en la que la base fundamental es separar una gran pieza de software en módulos, con el fin de hacer más fácil el desarrollo y mejorar su mantenimiento

Esta arquitectura permite distribuir físicamente los procesos y los datos en forma más eficiente, lo que en computación distribuida afecta directamente el tráfico de la red, reduciéndolo significativamente.

3.2.1 Características de la arquitectura cliente-servidor

La arquitectura cliente-servidor define de manera clara aspectos de comunicación, interacción y estructura que ayudan a la toma de decisiones durante el desarrollo. Las principales características aplicadas en la solución desarrollada se señalan a continuación:

- El proceso cliente proporciona la interfaz entre el usuario y el resto del sistema. Esta interfaz permite desplegar la información requerida y el manejo de eventos y solicitudes al servidor.

- El proceso servidor actúa como un administrador de software que maneja recursos compartidos como el modelo de persistencia de dato.

- Las tareas de lado del cliente y servidor tienen diferentes requerimientos en cuanto a recursos de cómputo como velocidad del procesador, memoria, capacidad del disco duro y dispositivos de entrada-salida.

- Se establece una relación entre procesos distintos, los cuales son ejecutados en equipos diferentes a través de una red.

- La relación establecida puede ser de muchos a uno, en la que el servidor puede dar servicio a muchos clientes, regulando el acceso a recursos compartidos.

- Los clientes corresponden a procesos activos en cuanto a que son éstos los que hacen peticiones de servicios a los servidores. Estos últimos tienen carácter pasivo ya que esperan las peticiones de los clientes.

- La única relación establecida entre cliente y servidor es la del intercambio de mensaje entre ambos. El mensaje es el mecanismo para la petición y entrega de solicitudes de servicio.

- El ambiente es heterogéneo. La plataforma de *hardware* y el sistema operativo del cliente y del servidor pueden no ser los mismos. Esto precisa una de las principales ventajas de la arquitectura. Permite conectar clientes y servidores independientemente de sus plataformas.

- Es posible aplicar el concepto de escalabilidad horizontal y vertical a cualquier sistema de la arquitectura. La escalabilidad horizontal permite agregar más estaciones de trabajo sin afectar significativamente el rendimiento. La escalabilidad vertical permite mejorar las características del servidor o agregar múltiples servidores.

3.3 Lado cliente

El cliente es el proceso que permite al usuario formular los requerimientos. Mediante un proceso o evento enviarlos al servidor, se suele referirse a este proceso con el término *front-end*.

El Cliente maneja todas las funciones relacionadas con la manipulación y despliegue de datos, por lo que están desarrollados sobre plataformas que permiten construir interfaces gráficas de usuario (GUI), además de acceder a los servicios distribuidos en cualquier parte de una red. Las funciones que lleva a cabo el proceso cliente se resumen en los siguientes puntos:

- Administrar la interfaz de usuario.
- Interactuar con el usuario.
- Procesar la lógica de la solución y hacer validaciones locales.
- Generar requerimientos de bases de datos.
- Recibir resultados del servidor.
- Dar formato a resultados.

El proceso cliente fue implementado siguiendo estándares de desarrollo, y patrones de arquitecturas adaptados a una aplicación Web. El patrón de diseño MVC (Modelo-Vista-Controlador) es la base de la arquitectura en este nivel de la solución, el cual contribuyo a la separación de los componentes relacionados con los datos y componentes de la interfaz de usuario. La separación de las capas permitió tener a nivel de desarrollo, código más claro, flexible y reusable.

En el caso de estudio de la solución desarrollada el Patrón MVC fue aplicado empleando frameworks de enmarcado y manejo de eventos. La estructura modelada establece una comunicación intrínseca entre las tres capas. A continuación se detalla cómo se representa cada capa del patrón en el entorno Web de la solución.

- Los archivos HTML de la solución representan la vista y debe ser separada del controlador y el modelo.

- El controlador es el encargado de preparar los datos a utilizar, capturar los eventos de la vista y realizar las acciones pertinentes sobre el modelo para modificar los datos. El controlador debe ser importado en el archivo de vista correspondiente para que este tenga efecto.

- El modelo es un objeto local nombrado como elemento del controlador $scope. Contiene los datos a los que va a acceder la vista y deberá contener también métodos de acceso y modificación para separar totalmente la forma a la que se acceden o modifican los datos del controlador. Adicionalmente el objeto $scope puede contener representaciones de elementos HTML, entidades de negocio, datos de control o cualquier unidad de información que se requiera configurar.

En la **Figura 10** se muestra el Patrón MVC adaptado a la tecnología empleada, así como también los diferentes componentes que interactuar y componen cada capa.

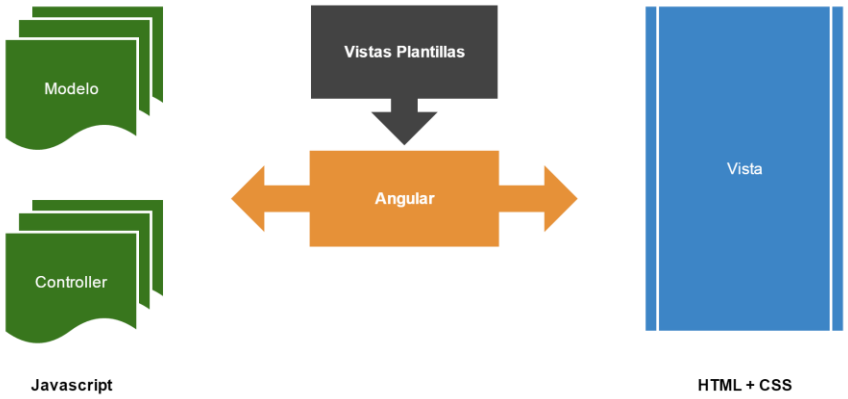


Figura 10 - Patrón de diseño MVC Aplicado al *framework* AngularJS.

3.4 Módulos funcionales de la solución del lado cliente

Como se mencionó anteriormente, el proceso cliente fue diseñado en capas que corresponden al patrón de diseño MVC. Adicionalmente, en cada capa se constituye de una serie de componentes que proveen ciertos servicios o funcionalidades que permiten controlar aspectos como la solicitud de recursos al servidor, estado de la solución, componentes Web, entre otros. En la **Figura 11**, puede observar los módulos más importantes.

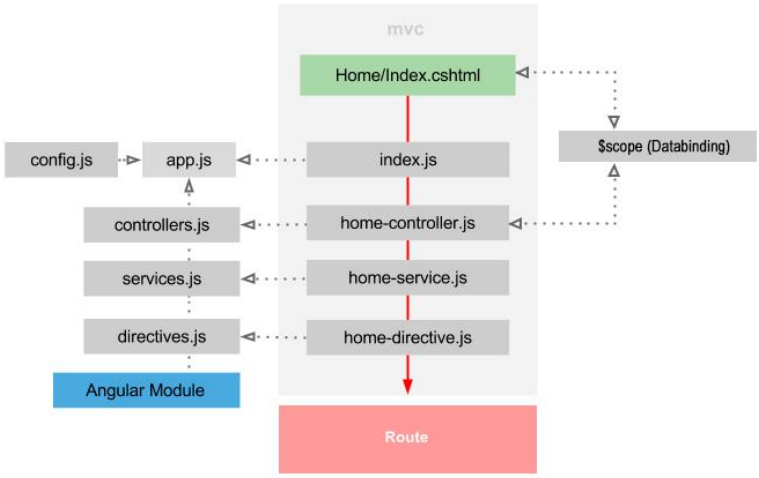


Figura 11 - Módulos funcionales de la solución del lado cliente.

- ***Angular Module***: Código base o *framework* que soporta a la solución. Puede ser vista como la biblioteca principal que contiene todas las rutinas necesarias para el correcto funcionamiento del sistema.

- ***Configuración (config.js)***: Contiene la información necesaria para adaptar la solución del lado del cliente a los requerimientos exigidos. En él se encuentra información susceptible como datos de conexión al servidor, protocolo a emplear, configuración de módulos externos (*add-ons*), entre otras.

- ***Aplicación (app.js)***: Módulo principal que permite enlazar y comunicar los diferentes elementos de la solución empleando el *framework* y la configuración anteriormente descrita.

- ***Servicios (Services)***: Permiten desarrollar funciones y objetos con propósitos específicos que posteriormente pueden ser incluidos en los controladores. En la factorías y servicios puede incluirse arreglos de datos, funciones e incluso otros objetos necesarios para modelar el comportamiento o funcionalidad necesario.

- ***Directivas (Directives)***: Permiten crear componentes Web con estructuras, estilo y comportamientos independiste. Se pueden incluir en cualquier vista a través de propiedades o etiquetas propias.

- ***Enlace de datos (Data Binding)***: Establece un mecanismo mediante el cual, de manera automática se actualiza la vista cuando ocurren cambios en el modelo, así como la actualización del modelo cada vez que cambia en la vista. Esto supone una ventaja en relación a la manipulación y actualización de elementos Web al generarse un cambio en los datos.

- ***Enrutador (Router)***: Módulo de enrutamiento, que permite organizar las partes de la interfaz de la solución en una máquina de estados. Se puede nombrar, enlazar vistas y agregar comportamientos a cada estado, lo que permite administrar eficientemente la interfaz de la solución.

3.5 Lado servidor

Es el proceso encargado de atender a múltiples clientes que hacen peticiones de algún recurso administrado por éste. Al proceso servidor se le conoce con el término *back-end*. El servidor normalmente maneja todas las funciones relacionadas con la mayoría de las reglas del negocio y los recursos de datos. Las funciones que lleva a cabo el proceso servidor se resumen en los siguientes puntos:

- Aceptar los requerimientos de bases de datos que hacen los clientes.
- Procesar requerimientos de bases de datos.
- Reiniciar y limpiar datos para trasmitirlos a los clientes.
- Procesar la lógica de la solución y realizar validaciones a nivel de bases de datos.

Como fase final en la estructura del proceso servidor, se complementa la arquitectura con la inclusión de un servidor de aplicación y base de datos. Basados en las características del sistema desarrollado era conveniente la incorporación de un servidor de aplicación que permitiera generar contenido de manera dinámica, y permitiera la toma de decisiones de acuerdo a parámetros de entrada incluidos en las peticiones por parte del cliente.

Finalmente, el servidor de base de datos está presente para apoyar de forma directa la persistencia de datos. Esta es requerida para almacenar y recuperar el estado de las entidades e información que pueda generarse a través de la solución. Todos los componentes descritos se muestran en la **Figura 12**.

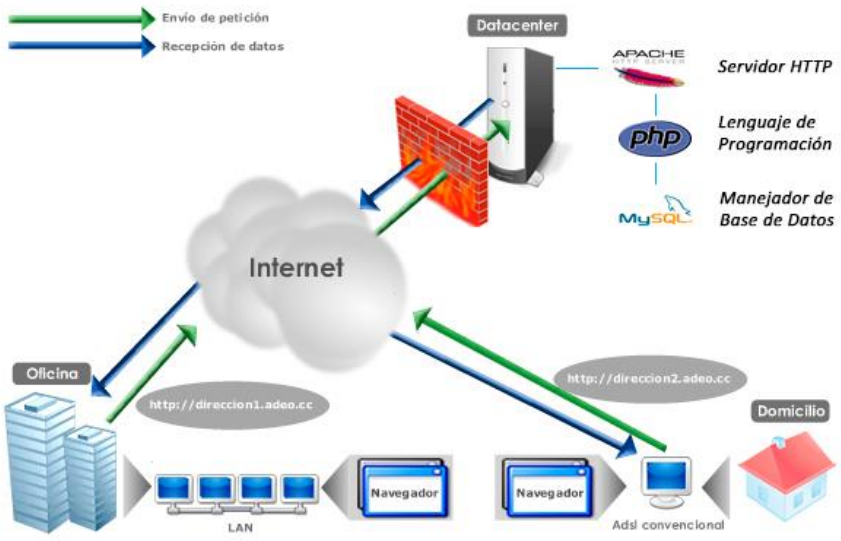


Figura 12 - Arquitectura de comunicación y software del lado servidor.

3.6 Módulos funcionales de la solución servidor

Para concebir la solución de forma abstracta y adaptable, se diseñó sobre el servidor de aplicación un sistema distribuido dividido en capas (Patrón de arquitectura multicapa). El objetivo principal fue separar los componentes de acuerdo a su función. Luego de un proceso de análisis se establecieron las siguientes capas funcionales: persistencia de datos, acceso a datos, entidades y lógica de negocio.

- ***Capa de persistencia de datos**:* Esta capa se basa directamente en el gestor de base de datos empleado. Se encarga de interactuar en el lenguaje apropiado con el *driver* de base de datos[13] y dar respuesta a las solicitudes realizadas por la capa de acceso a datos.

- ***Capa de acceso a datos**:* Proveer una interfaz que se encarga del acceso a los datos independientemente del tipo de gestor de base de datos que se utilice. De esta manera se puede incorporar otro gestor de base de datos a la solución sin afectar la estructura y funciones de las capas adyacentes.

- ***Capas de entidades**:* Al momento de establecen la comunicación entre la capa lógica de negocio y acceso a datos surge la necesidad de tener entidades o clases de dominio que representen a los elementos del negocio. De esta manera surgen las entidades, que proveen una representación única de cada recurso y unifica su acceso en cualquiera de las capas.

- ***Lógica de negocio**:* Se encarga de codificar las reglas de negocio de la solución para determina como la información puede ser creada, mostrada y cambiada. En esta capa están incluidas todas las rutinas que realizan entradas y consultas de datos, generación de informes y más específicamente todo el procesamiento que se realiza detrás de la solución visible para el usuario.

3.7 Servicio Web

El consorcio W3C define los Servicios Web como sistemas de software diseñados para soportar una interacción interoperable máquina a máquina sobre una red. Los Servicios Web suelen ser APIs Web que pueden ser accedidas dentro de una red (principalmente Internet) y son ejecutados en el sistema que los aloja.

La definición de Servicios Web propuesta puede albergar muchos tipos de sistemas, sin embargo, todas suelen referirse a la arquitectura clientes-servidores para establecen la comunicación mediante mensajes XML/JSON. En los últimos años se ha popularizado un estilo de arquitectura de software conocido como REST (*Representational State Transfer*). Este nuevo estilo ha supuesto una nueva opción de estilo en el uso de los Servicios Web.

3.7.1 Arquitectura REST

La lógica de diseño detrás de la arquitectura Web REST (*Representational State Tranfer*) puede ser descrita por un estilo arquitectónico que consiste en el conjunto de restricciones aplicadas a los elementos dentro de esta. De la misma manera, pretende maximiza la independencia y la escalabilidad de sus componente.

[13] Programas utilizados durante el tiempo de creación y procesamiento de las base de datos

REST está basado en estándares como HTTP, URL, representación de los recursos (XML, HTML, JPEG) y tipos MIME (text/html, text/xml). Este estilo de arquitectura de software es ampliamente usado en sistema hipermedia distribuidos tales como la Web. Los objetivos conseguidos al emplear la arquitectura REST en el desarrollo son las siguientes:

- Escalabilidad de la interacción de los componentes. Una amplia variedad de clientes pueden acceder a estos a través de la Web.

- Crear interfaces generales. Al emplear el uso del protocolo HTTP, cualquier cliente puede interactuar con el servidor HTTP sin configuraciones particulares.

- Puesta en funcionamiento independiente. Al tratarse de una solución accedida desde la Web, hay que tener en cuenta que servidores antiguos deben ser capaces de entenderse con clientes actuales y viceversa. HTTP permite la extensibilidad mediante el uso de las cabeceras, a través de las URIs[14], a través de la habilidad para crear nuevos métodos y tipos de contenido.

- Permitir la compatibilidad de los componentes internos, como tipos de proxys para la Web, módulo de cache y sistemas para reforzar las políticas de seguridad: firewall.

En la arquitectura desarrollada, los servicios no publican un conjunto arbitrario de métodos u operaciones, lo que publican son recursos y un recurso es considerado como la entidad que tiene un estado y un comportamiento, el cual puede ser accedido públicamente. Cada recurso dentro de la solución, posee un identificador único y global, que lo distingue de otros, aunque ambos tuvieran exactamente los mismos datos.

Todos los recursos comparten una interfaz única y constante, y tienen las mismas operaciones, las cuales permiten manipular el estado público del recurso. En el sistema REST implementado se emplearon sus 4 operaciones básicas:

- **Crear (*Create*)**: El cliente envía una petición al servidor junto a la representación de un recurso, para que este sea creado.
- **Eliminar (*Delete*)**: El cliente elimina un recurso del servidor. Este es frecuentemente indicado a través de su identificador único.
- **Leer (*Read*)**: El cliente solicita una representación del estado de un recurso.
- **Actualizar (*Update*)**: El cliente puede sobrescribir o grabar su copia del estado de un recurso en el servidor, actualizando el estado del recurso.

[14] Del inglés *Uniform Resource Identifier*, es una cadena de caracteres que identifica los recursos de una red de forma unívoca.

Estas acciones pueden llevarse a cabo mediante operaciones o comandos HTTP y empleando las URIs establecidas para cada recurso. Las URIs identifican y representan los recursos asociados al dominio de la solución. En la siguiente lista se indican los criterios aplicados en la creación de las URIs que conforman el API de la solución:

- ***Nombres cortos y precisos***: Esto hace que sean fáciles de escribir y recordar.

- ***Describen una jerarquía***: El usuario puede eliminar de la ruta una hoja (/example) y obtener una página posterior. Por ejemplo http://dv.dev/usuarios/1/modelos. Se podría eliminar la hoja "modelos" y aun así obtener los datos del recurso usuario con el identificador 1.

- ***Argumentos de configuración***: Es permisible adaptar la solicitud de un recurso empleando parámetros de configuración a través de *Query String*[15] (cadena de consulta) al final de la URI.

- ***Identificadores significativo y predecibles***: Se describe cada recurso al que se accede mediante el identificador del recurso. De esta manera luego de comprender la estructura planteada, los recursos posiblemente serán accedidos de manera natural por los usuarios o futuros desarrolladores.

- ***Representación uniforme***: Todos los recursos son representados mediante el formato de texto JSON.

En la **Tabla 6** se puede consultar la nomenclatura completa de URIs y acciones HTTP diseñada para dar acceso a los clientes a los diferentes recursos de la solución.

Verbo	Path	Acción	Nombre de ruta
GET	/resource	Index	Resource.index
POST	/resource	Store	Resource.store
GET	/resource/{resource}	Show	Resource.show
PUT/PATCH	/resource/{resource}	Update	Resource.update
DELETE	/resource/{resource}	Destroy	Resource.destroy

Tabla 6 - Acciones y URIs asociadas a un recurso.

Cada acción define operaciones particulares sobre el recurso involucrado. Para comprender como la arquitectura opera y como se representan los objetos dentro de la solución, a continuación se explica más detalladamente el recurso usuarios (aplicable a cualquier recurso) y nos basaremos en un nombre de dominio genérico (www.sv.dev) para ilustrar los URIs.

[15] Parte de una URL o URI, la cual contiene datos que deben ser transferidos a la aplicación Web.

- Obtener una copia del estado del recurso
  [GET: www.sv.dev/usuarios] Permite acceder al listado de los usuarios.
  [GET: www.sv.dev/usuarios/5] Permite acceder al detalle del usuario con el identificador 5.

  Cada petición puede ser configurada a través de la cadena de consulta. Los parámetros soportados permiten ajustar la cantidad de propiedades solicitadas por cada recurso (*data_length*), número de elementos por página (*page*, *per_page*) e incluso obtener la representación física del recurso en el servidor de base de datos (*dummy*).

- Crear instancias nuevas del recursos
  [POST: www.sv.dev/usuarios] Permite crear un nuevo usuario. Se debe enviar en la petición la información que se necesita para la creación en formato JSON: {nombre: "Luis", apellido: "Pérez", edad: 28}

- Actualizar el estado del recurso
  [PUT: www.sv.dev/usuarios/5] Permite editar el usuario con el identificador 5. Al igual que en la acción POST, se pasan los datos necesarios como cuerpo de la petición: {nombre: "Luisa", apellido: "De Pérez"}

- Eliminar un recurso
  [DELETE: www.sv.dev/usuarios/5] Permite eliminar el usuario con el identificador 5.

Para indicar el estado de cualquiera de las operaciones anteriormente descritas, la arquitectura utiliza la lista de códigos de respuesta del protocolo HTTP y las frases estándar asociadas, destinadas a dar una descripción corta del estatus. Los códigos más comunes son presentados en la **Tabla 7**.

Código	Mensaje	Código	Mensaje
200	Petición correcta	**400**	Solicitud incorrecta
201	Petición completada y recurso creado correctamente	***401***	No autorizado
202	Petición aceptada pero no completada	**404**	No encontrado
304	No modificado	**500**	Error interno del servidor

Tabla 7 - Códigos HTTP utilizados en la solución.

3.8 Patrones de Diseño de Software

Los patrones de diseño de software son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y también para otros ámbitos referentes al diseño de

interfaces. Los patrones empleados contribuyeron a estimar tiempo de desarrollo, detectar errores, dividir tareas y en planificar mantener el desarrollo en buen estado para su futuro mantenimiento y mejora.

De la misma manera, se emplearon diversas metodologías de desarrollo con el objetivo de estructurar, organizar y clasificar el contenido. Esto permitió establecer puntos de acceso, sistemas de búsqueda y de recuperación de la información de forma óptima y segura.

Durante el proceso de desarrollo se utilizaron patrones y enfoques adicionales para dar respuesta a problemas asociados a la naturaleza de la solución. Esto permitió establecer enfoques para el manejo de la calidad, confiabilidad, portabilidad y rendimiento del sistema. Al emplear patrones fue posible crear catálogos de elementos reusables, aplicar soluciones conocidas a problemas propios del sistema y dejar plasmado un lenguaje estándar que pueda ser reconocido fácilmente por próximos desarrolladores o diseñadores.

3.8.1 Patrones de diseños utilizados

De forma implícita o explícita fueron muchos los patrones de diseño empleados, sin embargo, se mencionaran a continuación aquellos que fueron utilizados con mayor frecuencia o representaron mejoras significativas en el resultado final.

- ***Fábrica Abstracta (Abstract Factory)***: Plantea una solución para creación objetos diferentes, pero pertenecientes a un mismo grupo o familia.

- ***Prototipo (Prototype)***: Tiene como finalidad crear nuevos objetos duplicándolos, clonando una instancia creada previamente. Este patrón especifica la clase de objetos a crear mediante la clonación de un prototipo que es una instancia ya creada. El lenguaje *Javascript* emplea este patrón ampliamente para el manejo de sus objetos.

- ***Constructor (Builder)***: Permite la creación de una variedad de objetos complejos desde un objeto fuente. Este objeto fuente, se compone de una variedad de partes que contribuyen a la creación de cada objeto complejo a través de llamadas o interfaces comunes a la clase constructora. Un caso aplicable es el manejo de la escena 3D. Por si sola una escena no es capaz de proveer ningún resultado, elementos complejos que la componen hacen posible el despliegue de una imagen.

- ***Instancia única (Singleton)***: Está diseñado para restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto. Su intención consiste en garantizar que una clase sólo tenga una instancia y proporcionar un punto de acceso global a ella.

3.2.3 Patrones de comportamiento utilizados

Los patrones de diseño de software de comportamiento son aquellos que están relacionados con algoritmos y con la asignación de responsabilidades a los objetos. Describen no solamente patrones de objetos o de clases, sino que también engloban patrones de comunicación entre ellos. Los siguientes patrones de comportamiento son aplicados ampliamente en el sistema desarrollado.

- ***Comando (Command)***: Permite realizar una operación sobre un objeto sin conocer realmente las instrucciones de esta operación ni el receptor real de la misma. Esto se consiguió encapsulando la petición como si fuera un objeto a través de los parámetros del ente ejecutor.

- ***Método de plantilla (Template Method)***: Se caracteriza por la definición de subclases que basan sus procesos a partir de una superclase. Estos procesos pueden redefinirse en la subclase de forma parcial o total.

- ***Observador (Observer)***: Permitió definir dependencias del tipo uno-a-muchos entre objetos, como por ejemplo los modelos de la aplicación y los diferentes eventos del sistema manejador de base de datos. De esta manera cuando uno de los objetos cambia su estado, notifica este cambio a todos los dependientes.

- ***Iterador (Iterator)***: define una interfaz que permitió declarar métodos para acceder de manera secuencialmente a las diferentes colecciones de objetos manipulados en la solución.

Capítulo 4 Diseño e Implementación

Para cumplir con la propuesta realizada se desarrollaron una seria de bibliotecas y clases (*RESTful*, *Scene, Chameleon,* entre otras) en los lenguajes PHP v5.4 y Javascript, las cuales fueron divididas en dos niveles (cliente-servidor), y contienen los métodos y propiedades necesarios para el funcionamiento de la solución *Shader Tool*.

La biblioteca creada del lado servidor se encarga de procesar las peticiones recibidas, y prepara el entorno para atenderla y dar respuesta, entre otras funciones. Del lado del cliente las clases desarrolladas se dividen en tres grupos principales. El primero de ellos es el encargado de gestionar las peticiones y respuestas obtenidas, el segundo realiza todas las tareas involucradas en el proceso de despliegue de la escena 3D, y finalmente, el último grupo provee componentes Web que encapsular funcionalidades y manejo de eventos en la interfaz de usuario.

4.1 Aspectos técnicos

Para la implementación de la solución, bibliotecas y clases se utilizó el entorno Web, bajo la arquitectura cliente-servidor. El sistema se puede acceder desde cualquier ambiente de sistema operativo (Unix o Windows) a través de un navegador Web. En la **Tabla 8** se puede ver el soporte que tiene la solución en los principales navegadores comerciales.

	IE8	IE9 o superior	Firefox	Chrome	Safari
AngularJS 1.3.8	✖	✔	✔	✔	✔

Tabla 8 - Soporte de AngularJS en navegadores actuales.

El sistema en su totalidad está conformado por dos subsistemas, los cuales operan en cada nivel de la arquitectura. Del lado del cliente se utilizó el lenguaje *JavaScript* y diversas bibliotecas de licencia libre como: jQuery, AngularJS, ThreeJS, entre otras.

> ***JavaScript***
> *Es un lenguaje de programación interpretado empleado en el diseño de sitios Web. Permite crear esquemas de programación estructurada, tipos de datos dinámicos, evaluaciones en tiempo de ejecución, modalidad de programación funcional, prototipos, definición literal de objetos, entre otras.*

Para el desarrollo de interfaces se utilizó el lenguaje HTML para enmarcar el contenido, y CSS para mejorar el aspecto visual de cada vista. Para complementar la fase de desarrollo de las

interfaces, se incluyó el *Frameworks de CSS* Bootstrap y herramientas de pre procesamiento de hojas de estilo *LESS*[16].

> ***HTML y CSS***
> *HTML es un lenguaje de marcado para la elaboración de páginas Web. Define una estructura básica para la definición de contenido de una página Web, como texto, imágenes, videos, entre otros. Por otra parte, CSS se utilizar para adaptar el aspecto grafico de estas páginas y la forma en que son presentadas en el navegador Web.*

Del lado de servidor el lenguaje empleado fue PHP 5.4, sobre el cual está diseñado el *framework* Laravel 4.2, y los componentes adicionales que se incorporaron como *AuthToken* (manejo de sesiones), *Generator* (utilidades de generación de archivos en fase de desarrollo), *FileUpload* (Gestor de archivos recibidos desde la solución cliente), *Chameleon* (Utilidades varias para la implementación de la lógica de negocio). Tanto el *framework* como los componentes descritos permitieron el desarrollo de la API REST en su versión 0.1.0 que brinda soporte a la solución cliente.

> ***PHP***
> *Este procesador de hypertexto (Hypertext Preprocessor) es un lenguaje de código abierto adecuado para el desarrollo Web y que puede ser incrustado en documentos HTML de forma dinamica.*

Del lado del servidor, se utilizó el sistema de gestión de base de datos MySQL en su versión 5.6.17. En él se modelaron todos los recursos involucrados en la solución y objetos adicionales que permiten manejar a los usuarios conectados y su configuración. Para generar estos objetos de datos se utilizó la herramienta *Artisan*, la cual es una interfaz de línea de comando que por medio de archivos de configuración permite crear y gestionar dichos objetos de datos.

> ***MySQL***
> *Sistema de administración de bases de datos (Database Management System, DBMS) para bases de datos relacionales, multihilo y multiusuario.*

En el desarrollo del sistema (ambos niveles) se utilizó el paradigma de Programación Orientada a Objetos[17] haciendo uso de ciertas convenciones en la nominación de las clases, nombres

[16] LESS es un pre procesador de CSS. Añade características que permiten el uso de variables, funciones, entre otras técnicas, que permiten realizar CSS de una manera más fácil de mantener.

de bibliotecas, estructuras, atributos, métodos y constantes. Los nombres de clases, bibliotecas, métodos, constantes y atributos públicos siempre comienzan con la primera letra en mayúscula. La nominación de los atributos privados contiene el prefijo "_" con la primera letra en minúscula. Los nombres de las variables locales y parámetros comienzan en minúsculas sin prefijo alguno. Los nombres de las clases empleadas como controladores contienen el sufijo "*Controller*".

En todos los casos si los nombres están formados por dos o más palabras se utiliza la notación *UpperCamelCase* (Mayúsculas/Minúscula Camello). Por ejemplo: *UsersController*, *Shaders, extendedBeforeFilter(), _isActive, ShaderMaterial, data,* etc.

En relación al servidor HTTP, se utilizó durante la fase de desarrollo el servidor HTTP Apache en su versión 2.4.9 de código abierto para el sistema operativo Windows 8 de 64 bits. El software está disponible para diversas plataformas de Microsoft Windows, Unix (BSD, GNU/Linux, entre otros), Macintosh y otros. Los módulos de Apache utilizados son tres: *mod_headers* para manipular las cabeceras HTTP mediante la directriz Header, *mod_mime* el cual entrega a los clientes meta-información sobre los documentos y *mod_rewrite* que reescribe las URL para que tengan una estructura más adecuada.

4.2 Arquitectura del API REST

REST (*Representational State Transfer*) es un estilo de arquitectura de software para sistemas hipermedias distribuidos tales como la Web. REST se refiere estrictamente a una colección de principios para el diseño de arquitecturas en red. Estos principios resumen como los recursos son definidos y diseccionados.

La API REST de la solución fue construida utilizando el *framework* Laravel en su versión 4.2. Es de código abierto y trabaja con el lenguaje de programación PHP. Sus características principales son un sistema de ruteo (también *RESTful*[18]), *Eloquent ORM*, Basado en *Composer*[19], soporte para el caché y finalmente soporte para el patrón de arquitectura MVC.

La solución que define la API está formada por tres módulos/clases principales: *Database*, *Routing* y *Controller*. El objetivo de dividir la solución servidor en módulos es hacerla más legible, manejable y de fácil mantenimiento. Cada uno de los módulos del sistema representa una estructura lógica con tareas y opciones específicas. Dichos módulos se pueden comunicar entre ellos a través de interfaces bien definidas, tales como clases, métodos y atributos.

---

[17] La programación orientada a objetos o POO (OOP según sus siglas en inglés) es un paradigma de programación que usa los objetos en sus interacciones. Está basado en varias técnicas, incluyendo herencia, cohesión, abstracción, polimorfismo, acoplamiento y encapsulamiento.

[18] Conjunto de servicios implementados bajo la arquitectura REST

[19] *Composer* es una herramienta para administración de dependencias en PHP. Permite declarar las bibliotecas de las cuales el proyecto depende o necesita y éste las integra al proyecto.

Adicionalmente, se hace uso de dos componentes externos llamados *Laravel Auth Token* y *Laravel File Upload,* los cuales facilitan el manejo de la autenticación basada en *Token* y la manipulación de archivos digitales, respectivamente. En la **Figura 13** se describe la relación que hay entre los diferentes módulos.

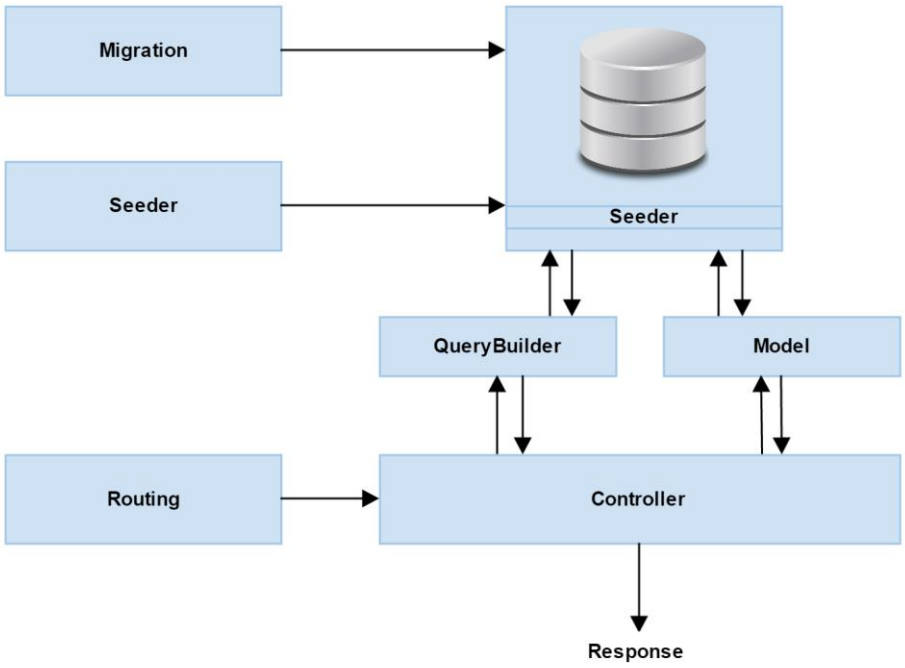


Figura 13 - Módulos de la arquitectura de la API REST.

Es importante indicar que el framework provee módulos adicionales, sin embargo, los mostrados en el **Figura 13** son los más relevantes en el desarrollo de la solución.

4.3 Módulo *Database*

En el módulo de base de datos se encuentran todos los métodos y clases necesarias para el manejo del esquema de base de datos, las funciones para crear o modificar las tablas con la que opera la solución servidor y los métodos para incluir las semillas de datos en cada una de ellas. El módulo está conformado por los paquetes *Eloquent, Migration* y *Seeder*.

4.3.1 Paquete *Eloquent*

Dentro de la estructura de la solución servidor cada tabla está representada por una clase que provee las interfaces necesarias para manipularla, crear las relaciones lógicas, establecer la conexión con la base de datos, entre otras. El paquete está representado por una clase principal nombrada *Model que es la encargada de encapsular todas las funcionalidades.*

La clase *Model* fue diseñada empleando el patrón de diseño *Composite* y está constituida por elementos más simples, entre las cuales se encuentran: *Dispatcher, Relations* y *QueryBuilder.* Aunque intervienen otras clases y paquetes, estas son que contribuyen mayormente al objetivo principal del paquete, o están presente de forma más activa. En la **Figura 14** se muestra el diagrama de clase simplificado del paquete.

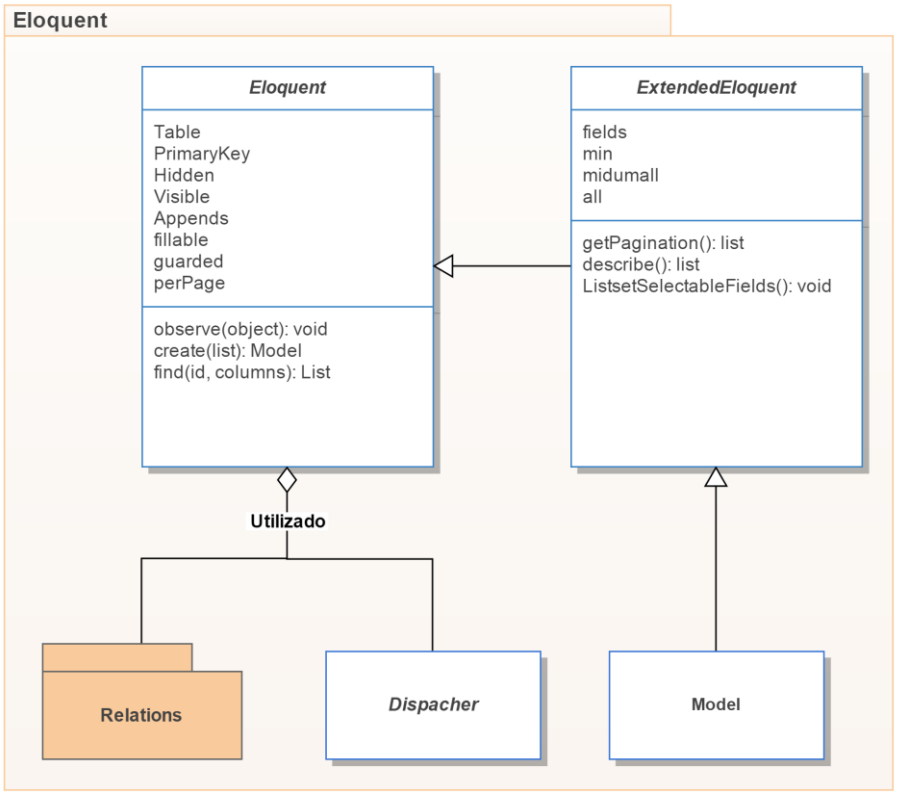


Figura 14 - Diagrama de clases del paquete *Eloquent*.

Las propiedades más importantes utilizadas en la clase modelo son los siguientes (ver **Tabla 9**):

Método	Descripción
$Table	Representa la tabla asociada al modelo.
$primaryKey	Clave primaria del modelo
$Hidden	Permite limitar los atributos que son incluidos en el objeto JSON del modelo, como contraseñas
$Visible	Alternativamente, se puede utilizar la propiedad visible para definir una lista blanca.
$appends	Permite añadir atributos que no corresponden a una columna en la base de datos al arreglo de atributos del modelo.

Tabla 9 - Atributos importantes de la clase *Model*.

Adicionalmente estas propiedades son complementadas por la clase padre *Extended Eloquent*, que incorpora atributos adicionales para gestionar de forma dinámica algunas propiedades propias de las peticiones, entre las cuales se encuentra la cantidad de atributos del recurso solicitado y la descripción del esquema de base de dato de alguna tabla.

4.3.1.1 Clase *Dispacher*

Los modelos *Eloquent* disparan varios eventos, permitiendo así actuar en varios puntos del ciclo de vida del modelo utilizando los siguientes métodos: *creating* (creando), *created* (creado), *updating* (actualizando), *updated* (actualizado), *saving* (guardando), *saved* (guardado), *deleting* (eliminando), *deleted* (eliminado), *restoring* (restaurando), *restored* (restaurado).

Cada vez que un nuevo elemento es guardado por primera vez, serán disparados los eventos *creating* y *created*. Si un elemento no es nuevo pero se llama al método *save*, los eventos *updating* / *updated* se dispararán. En ambos casos, los eventos *saving* / *saved* se habrán disparado. Para cada evento se puede asociar una función (*callback*) en cada modelo, donde se pueden realizar operaciones o validación del proceso.

Los eventos *creating, updating, saving y deleting* permiten cancelar la culminación de la operación si la función *callback* retorna un valor *false*. La forma básica del manejo de eventos se puede observar en el **Código 1**.

```
/* Register event callback */
Model::eventName(function($modelObject){
   if (!$ modelObject ->isValid()) return false;
});
```

Código 1 - Manejo de eventos del Modelo.

Para consolidar el manejo de eventos en el modelo, se puede registrar un observador de modelo. Una clase "*observer*" puede tener métodos que corresponden a los diferentes eventos del modelo. Por ejemplo, los métodos *creating*, *updating*, *saving* pueden alojarse en un *observer*, además de cualquier otro evento del modelo. Así, por ejemplo, un observador de modelo podría ser (ver **Código 2**):

```
/* Create observer */
class ModelObserver {
   public function saving($model){ ... }
   public function saved($model){ ... }
}
/* Register observer */
Model::observe(new ModelObserver);
```

Código 2 - Registro de observador de eventos de Modelo.

4.3.1.2 Paquete *Relations*

El paquete permite crear relaciones lógicas entre los modelos de la solución del lado servidor. Los tipos de relaciones soportadas son las conocidas en los modelos de datos relacionales: uno a uno, uno a muchos, muchos a muchos, muchos a través de, entre otras.

Las relaciones son representadas por clases que están directamente relacionadas (patrón de diseño *Composite* y *Contructor*): *Relation* (clase base), *HasOne*, *HasMany*, *BelongsToMany*, *HasOneOrMany* y *BelongsTo*. En la **Figura 15** se muestra el diagrama de clase del paquete.

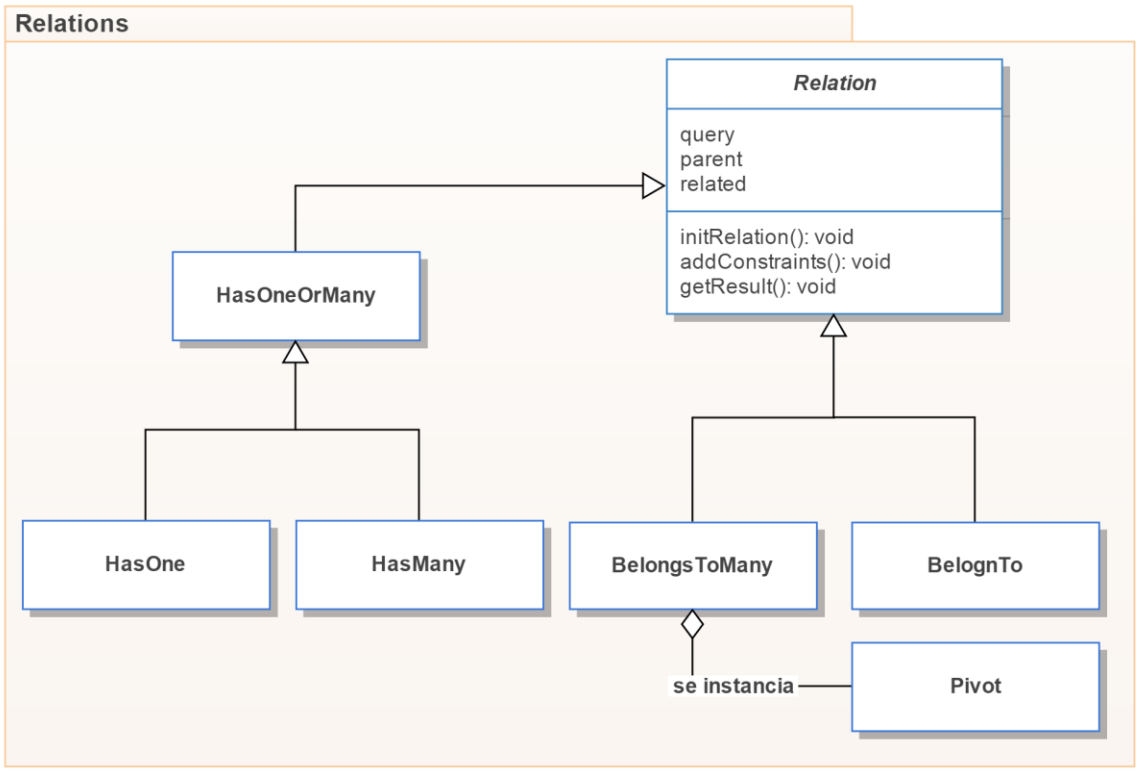


Figura 15 - Diagrama de clases del paquete *Relations*.

La clase *Relation* define los atributos y métodos básicos para establecer una relación entre dos modelos. Al tratarse de una clase abstracta[20], algunos de sus funciones deben ser definidos en la clase hija, para así permitir una relación en particular. La clase *Relation* posee atributos como *parent* y *related* los cuales identifican a los modelos involucrados en la relación, y *query* que posee la información necesaria para realizar una consulta al modelo de datos.

[20] Clase que declara la existencia de métodos pero no la implementación de dichos métodos.

La siguiente lista (ver **Tabla 10**) muestra los métodos abstractos principales de la interfaz:

Método	Descripción
Abstract initRelation()	Métodos que permiten inicializar los parámetros de configuración de la relación.
Abstract addConstraints()	Permite registrar restricciones a la consulta final, permitiendo aplicar un grupo de restricciones a todas las consultas hechas por el modelo.
Abstract getResults()	Permite a la clase hija hacer el manejo de los resultados de la consulta de acuerdo al tipo de relación que implementa.

Tabla 10 - Métodos abstractos de la clase *Relation*.

Empleando la clase antes descrita, es posible establecer relaciones de uno a muchos y de muchos a muchos, a través de las clases *HasOne* y *HasMany* respectivamente. En ellas se defines los métodos abstractos declarados en la clase padre (*Relation)*. Es importante destacar que esta herencia no ocurre de forma directa, ya que estas extienden sus propiedades de una clase abstracta intermedia nombrada *HasOneOrMany*.

La relación de muchos-a-muchos esta modelada con interfaces adicionales a la clase base *Relation*, entre las cuales está la clase *Pivot*. Esta última permite crear de manera independiente y automática el modelo que representa la tabla derivada de este tipo de relación. De esta manera la relación aprovecha las características que ofrece un modelo (*Model*) para manipular los datos asociados.

Finalmente, el paquete permite a los modelos involucrados en una relación, definir su asociación en la dirección contraria, por tanto es posible tener acceso a la información de ambas tablas asociados desde cualquiera de ellas. Este tipo de relación es nombrada *BelongsTo*.

4.3.1.3 Paquete *QueryBuilder*

El generador de consultas de bases de datos (*QueryBuilder*) proporciona una interfaz cómoda para crear y ejecutar consultas de bases de datos. Se utilizar para realizar todas las operaciones de base de datos. El generador de consultas Laravel utiliza la extensión DOP[21] en todas las operaciones de base de datos para proteger la solución contra posibles ataques de inyección SQL. De modo que no hay necesidad de limpiar las cadenas provenientes del lado cliente.

La cadena de consulta es construida a partir de un objeto *Model* que es obtenido a través del método estático *table* del objeto *BD*. La forma de acceso puede verse en la **Código 3**:

[21] Extensión de PHP (Objetos de Datos) la cual define una interfaz ligera para poder acceder a bases de datos.

```php
/* Get all records in a table */
$records = DB::table('tableName')->get();

foreach ($records as $record){
   var_dump($user->field);

}
```

Código 3 - *Query* que da acceso a la lista completa de usuarios a través del módulo *QueryBuilder*.

Al tener la instancia del modelo solicitado, progresivamente se agregar condiciones a la cadena de búsqueda. Mediante su uso se pueden crear consultas selectivas complejas, uniones, inserciones, etc. Los métodos principales que proveen estas funcionalidades se muestran en la **Tabla 11**.

Método	Descripción
Get()	Procesa la consulta y retorna los registros. Cada uno de ellos, Estos vienen representados
Select()	Permite indicar las columnas a seleccionar.
Lists()	Restringe la respuesta solo una columna. Este método resulta útil al crear elementos HTML de selección (*select*) con listas de datos provenientes de una tabla.
Where()	Definir una condición en la cadena de consulta. El método recibe como parámetros la columna involucrada, el operador y el valor.
orWhere()	Registra una segunda clausula where aplicando el operador lógico or entre ambas.
whereBetween()	Permite acotar una clausula where a un rango de valores fácilmente. De este modo no es necesario crear dos condiciones.
groupBy ()	Presenta el contenido de la consulta agrupado por una columna existente en el objeto.
Join()	Extienden la cantidad de atributos por registros al crear una cláusula de unión con una segunda. Los parámetros de entrada de la función son: Tabla a unir y las columnas de ambas tablas con las cuales se hará la asociación entre sus registros.

Tabla 11 - Métodos disponibles

De igual modo, existen métodos avanzados que proveen herramientas para personalizar y describir a profundidad la cadena de consulta que se desea ejecutar. De ser necesario, es posible mediante el uso de funciones *callback* manipular directamente el objeto *$query* del modelo.

4.3.2 Paquete *Migrations*

Las migraciones son un tipo de control de versiones para la base de datos. Permite modificar el esquema de base de datos[22] y estar al día sobre el estado actual del esquema. La clase *Migration* es abstracta y define el tipo de conexión de base de datos usada. Dentro de la solución cada tabla está representada por una clase que hereda de la clase *Migration*, e internamente hacen uso de dos clases adicionales que son las que finalmente dan acceden al esquema de base de datos, *Blueprint* y *Schema*. Para ejemplificar, en el **Código 4** se define la clase que incluye la tabla usuarios al esquema de base de datos de la solución del lado servidor.

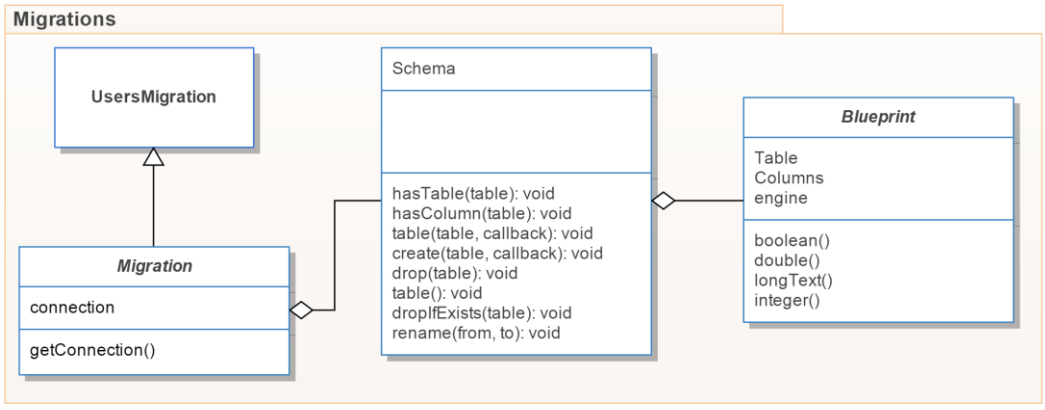
```
/* Create user table structure */
class CreateUsersTable{

    public function up(){
        Schema::create('users', function(Blueprint $table){ ... });
    }

    public function down(){
        Schema::drop('users');
    }
}
```

Código 4 - Clase que crear la tabla User en el esquema de base de datos.

La clase *Blueprint* permite definir las propiedades necesarias para crear una nueva tabla o representar una que pertenezca al esquema. Sus atributos más importantes son *$table* que describe el nombre de la tabla y *$columns* que defines los nombres de los campos que la componen. Adicionalmente provee métodos que permiten definir atributos como claves primarias, foráneas o columnas auto incremental. En la **Figura 16** se muestra el diagrama de clase del paquete.



[22] Describe la estructura de una base de datos, en un lenguaje formal soportado por un sistema de gestión de base de datos (DBMS).

Figura 16 - Diagrama de clases del paquete *Migrations*.

La *Clase Schema* expone métodos estáticos que permiten aplicar operaciones sobre una instancia de la clase *Blueprint* y es la que finalmente se encarga de traducir sus propiedades en una representación física dentro del esquema de base de datos. Esta clase surge como un enlace al paquete *QueryBuilder* y fue implementado bajo el patrón de diseño fachada (*Facade*). De esta manera se cuenta con una interfaz simple que da acceso a las funciones de interacción con el esquema de base de datos. Entre los métodos públicos más importantes de la clase *Schema* se encuentran (ver **Tabla 12**):

Método	Descripción
Create()	Crea una nueva tabla dentro del esquema de base de datos. Recibe como entrada la referencia de una clase *Blueprint* y una función callback que será invocada luego de la operación.
Drop()	Elimina una tabla del esquema de base de datos. Recibe como entrada la referencia de una clase *Blueprint.*
Table()	Permite modificar los atributos de una tabla. Al igual que en el método *Create()* sus parámetros de entradas con una instancia *Blueprint* y una función callback.

Tabla 12 - Métodos principales de la clase *Schema*.

4.3.3 Paquete *Seeder*

Los archivos semillas permiten configurar los datos de prueba y producción con los que contara la solución. Para cada tabla se creó una clase que contiene los registros bases que serán agregados, las funciones principales vienen dada de la clase padre *Seeder*. La forma de las clases semillas se muestran en el **Código 5**.

```
class « name »TableSeeder extends Seeder {
  public function run(){
    «Model»::create(array('field' => 'value'));
  }
}
```

Código 5 - Plantilla para la creación de semillas asociadas a un modelo.

El método principal del paquete es *run() y* debe ser implementado desde la clase hija. Este método será invocado en procesos posteriores para agregar los registros a la tabla correspondiente. Dentro del cuerpo de ese método se utiliza la clase DB como interfaz de acceso a la tabla y permite insertar los datos.

4.4 Módulo *Routing*

El módulo de enrutamiento es uno de los más elementales que conforman la solución del lado servidor, sin embargo, es de gran utilidad ya que permite ordenar los diferentes recursos en una estructura jerárquica, donde lógicamente se podría modelar la dependencia entre modelos y representarla a través de grupos de rutas.

Dentro de la solución servidor el proceso rounting está representado enteramente en el archivo app/route.php. Esté contiene todas las directrices necesarias que dan acceso a los recursos que ofrece la API. Estas directrices especifican URIs y en algunos casos parámetros que deben ser provistos para el acceso.

Adicionalmente, se encuentra el paquete *Request,* el cual es parte primordial del módulo. Se encarga de resumir en un único objeto toda la información relacionada con una petición. Este paquete permite acceder a atributos como el método utilizado en la petición, la dirección IP del cliente, el url completo, sus parámetros, entre otros.

Finalmente, existen otras clases más simples que se encargan de procesos más puntuales dentro del proceso de enrutamiento. *UriValidator* por su parte implementa un método que permite validar las URIs, *MethodValidator* comprueba si el método HTTP recibido tenga soporte y *SchemeValidator* comprueba que se utilice HTTPS cuando así se configure dentro del API REST.

4.4.1 Clase *Route*

Route provee los métodos necesarios para configurar las URIs que serán usadas desde el lado cliente para solicitar los recursos. Adicionalmente, se emplea el uso de grupos de rutas con el objetivo de versionar la API y aplicar filtros a varias rutas de manera conjunta.

Las URIs del API fueron agrupadas con el prefijo “api/0.1.0”. Para definir un recurso de manera completa se usa el método *Route::resource*. Esto permite automáticamente crear 4 URIs asociadas a un recurso. Para su configuración recibe dos parámetros, el nombre del recurso con el cual se quiere formar la URI y el controlador encargado de atender dichas peticiones. En la **Tabla 13** se muestran los 5 servicios que se generan con la instrucción tomando como ejemplo el recurso *Users*.

URIs	Método asociado	Descripción
GET: Example.com/users	*Index*	Da acceso a la lista de usuarios. Las restricciones asociadas al recurso son aplicadas internamente en el método.
GET: Example.com/users/{id}	*show*	Recibe por parámetro el id del usuario solicitado.

POST: Example.com/users	*Store*	Permite crear un nuevo recurso usuario.
PUT: Example.com/users/{id}	*Update*	Actualiza los datos de un usuario dado su identificador.
DELETE: Example.com/users/{id}	*Destroy*	Elimina un recurso usuario dado su identificador.

Tabla 13 - Lista de URIs generados automáticamente para un recurso (::*resource*).

Para algunos recursos se requirió configurar URIs adicionales para dar servicio a otras formas de presentación de los recursos. Los métodos utilizados fueron *get*, *post*, *put* y *delete*. Los parámetros que permiten en todos los casos son el nombre que identifica el recurso y define la cadena de la URI y el método perteneciente a un controlador que atiene la petición. Es importante resaltar que los métodos descritos hacen referencia a la acción HTTP a la cual se quiera el servicio.

4.4.2 Paquete *Request*

El paquete está representado por la clase *Request*. Recopila toda la información generada por una petición realizada desde la solución del lado cliente. Este objeto y sus componentes pueden ser accedidos desde cualquier parte de la solución servidor. Los dos atributos que define son:

4.4.2.1 *ParameterBag*

Los API REST se caracterizan por representar los recursos en un formato de dato manipulable. En la solución desarrollada la información viaja utilizando la representación de datos JSON. Este formato es apropiado para el desarrollo por su versatilidad y amplio soporte en los lenguajes de programación Web. La solución del lado servidor al recibir los datos los representa en una estructura genérica.

Los campos enviados desde el cliente (por ejemplo, datos de registro de usuario) son almacenados en la estructura *ParameterBag,* y son representados por medio de un diccionario clave-valor. La clase ofrece funciones para la gestión y manipulación de estos parámetros. Es importante indicar, que en el este proceso se deja a un lado el formato JSON y todos los campos son almacenados como tipos de datos de PHP. Los principales métodos que posee la clase *ParameterBag* se muestran en la **Tabla 14**:

URIs	Descripción
Get()	Retorna un parámetro dado su nombre.
Has()	Permite verificar la existencia de un campo.
Remove()	Elimina el parámetro dado su nombre.
Count()	Retorna la cantidad de parámetros incluidos en la petición.

Tabla 14 - Métodos principales de la clase *ParameterBag*.

4.4.2.2 *SessionStore*

El manejo de sesiones provee una manera de almacenar información sobre el usuario a través de las solicitudes. El framework provee soporte a esta característica por medio de la clase Store. Los mecanismos Memcached[23] y Redis[24] fueron implementados y están disponibles desde la clase.

La sesión es creada al momento que un usuario accede a la solución y la primera petición es recibida en el servidor Web. Los datos generados y almacenados son identificados por un ID, el cual es enviado en la respuesta al navegador Web. Este ID será enviado en cada petición y de esta manera la sesión pueda ser recuperada y usada.

La clase provee los atributos y métodos para que el mecanismo explicado anteriormente se lleve a cabo de forma transparente a la solución. La sesión puede ser accedida de varias maneras dentro de la estructura del *framework*, a través de métodos estáticos, clases *helpers* o mediante el objeto *Request*.

Los métodos para el manejo de atributos de sesión son los siguientes (ver **Tabla 15**):

Método	Descripción
Put()	Almacena un elemento en la sesión.
Push()	Añadir un nuevo valor a un arreglo de valores de sesión.
Get('key')	Provee el valor de un ítem previamente almacenado.
Pull('key')	Provee el valor de un ítem previamente almacenado y posteriormente se elimina de la sesión.
Forget('key')	Elimina un elemento de sesión.
Regenerate()	Genera un nuevo ID de sesión

Tabla 15 - Métodos disponibles para manejar atributos de sesión.

4.5 Módulo Controller

Los controladores se utilizan para organizar la lógica de la solución del lado servidor. Mediante clases Controller se pueden agrupar peticiones HTTP relacionadas. Dentro de la estructura las clases controladores heredan de la clase *ExtendedController*, la cual es una biblioteca que se desarrolló para encapsular funciones asociadas a los controladores y proveer una interfaz genérica.

[23] Mecanismo empleado para el almacenamiento en caché de datos u objetos en la memoria RAM, reduciendo así las necesidades de acceso a un origen de datos externo.
[24] Motor de base de datos en memoria, basado en el almacenamiento en tablas de hashes (clave/valor).

Cada identificador de recurso uniforme está compuesto por una cadena que lo define, un método HTTP y una función perteneciente a un controlador. La forma básica de todos los controladores de la solución se describen en el **Código 6**.

```
class « name »Controller extends ExtendedController{
  public function _contructor(Model, RulesValidation){ ... }
}
```

Código 6 - Plantilla que define el controlador de un recurso.

Como se mencionó anteriormente, el controlador es asociado a un recurso en el módulo de *Routing*. Este debe definir los métodos estándar para dar respuesta a las peticiones básicas de un recurso. Adicionales fueron incluidos en algunos controladores como *Models*Controllers, *ShadersController* y *TexturesController* métodos adicionales para ofrecer servicios especiales. De esta manera se facilita el acceso a un sub conjunto de los datos. Estos nuevos servicios son utilizados en gran parte para proveer de plantillas o archivos base a la solución cliente. En el diagrama de controladores (ver **Figura 17**) se muestra detalladamente todas las clases *Controllers* que conforma el API junto a sus atributos y métodos.

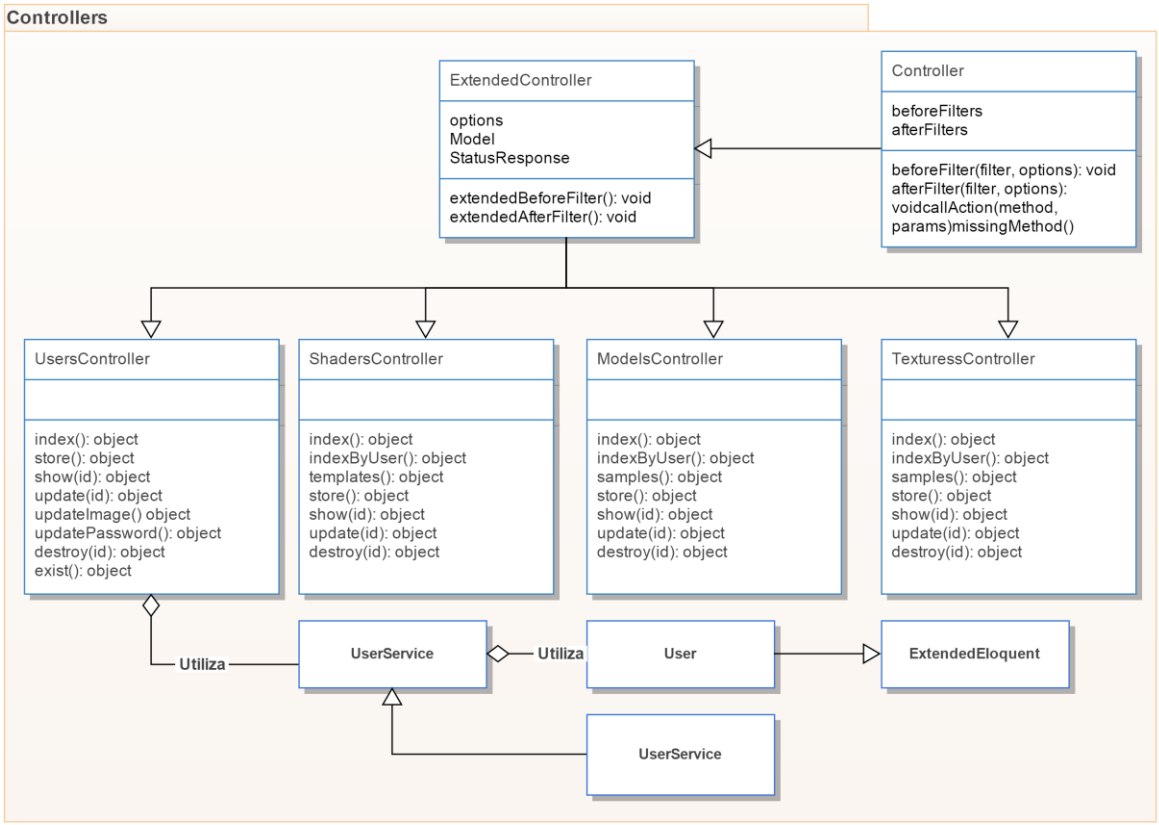


Figura 17 - Diagrama de clases del paquete *Controllers*.

4.5.1 Clase *ExtendedController*

ExtenderController es la clase principal que da forma a cada función dentro de cualquier controlador. Está compuesta por el controlador base (*BaseController*) que provee Laravel. Esta extensión a las funciones básicas de los controladores permite manejar de forma genérica el manejo de algunos filtros de gran importancia que son ejecutados antes y después de ejecutarse la función asociada a una petición, estos son conocidos como *BeforeFilters* y *AfterFilter* respectivamente. Los filtros habilitados dentro de cada controlador son tres, los cuales se describen a continuación.

4.5.1.1 *Dummy*

Este filtro se ejecuta antes de atender la petición y de acuerdo a ciertas validaciones es posible que el flujo normal de la petición sea interrumpido. Esta función permite solicitar una descripción completa de la tabla asociada al recurso, a través de la cadena de consulta de la URI.

Cuando se incluye el campo *dummy* con un valor verdadero (dummy=1) el filtro lo captura, detiene el flujo de la petición hasta ese punto y envía como respuesta dicha estructura. Por ejemplo, para el recurso usuario esta sería la representación del recurso (ver **Código 7**)

```
data: {
   table: "table_name",
   rows: [{
      Field: "field_name",
      Type: "data_type",
      Null: "NO/YES",
      Key: "{EMPTY,PRI}",
      Default: “value”
   }]
}
```

Código 7 - Objeto JSON de la estructura descriptiva de una tabla.

4.5.1.2 Data Lenght

Este filtro permite definir la cantidad de atributos que se requiere sean considerados en la respuesta del recurso solicitado. En ocasiones es necesario solicitar la información de un recurso por partes para así obtener los datos de forma más rápido. De esta manera, usando el campo data_lenght como parte de la cadena de consulta es posible indicar si se requiere el recurso de forma simplificada (*min*), normal (*medium*) o completa (*all*).

Cada Modelo especifica los atributos que pertenecen a cada grupo. Al recibir una petición este valor permite seleccionar dicho atributo al momento de realizar la consulta en la base de datos.

4.5.1.3 Status Response

Este filtro se ejecuta luego de atender la petición. Hace uso de ciertos datos recopilados en el proceso y otros provistos por el framework. Se crear un objeto que describe el estado de la petición. El objetivo es incluir de manera automática información de control (metadata) que permita desde la solución cliente evaluar el estado de la petición. El objeto creado tiene los siguientes atributos (ver **Tabla 16**)

Atributo	Descripción
HTTP Code	Hace referencia a los códigos establecidos en el protocolo de comunicación http.
Status	Indica si la operación fue procesada exitosamente (*success*), o si por el contrario no pudo ser completada (*error*).
Resource	Especifica el recurso involucrado en la petición.
Message	Mensaje que describe el estado de la petición. En los casos cuando la petición finaliza en error, este mensaje puede contener las causas, por ejemplo validaciones no aprobadas en los atributos de un nuevo recurso.
Url	URI completo de la petición.

Tabla 16 - Atributos configurables para el objeto de estado de la petición.

4.6 Descripción completa de la API v0.1.0

Las API REST proporcionan acceso programático a leer y escribir datos *Shader Tool*. Crear cuentas de usuario, añadir texturas, consultar programas de *shaders*, y más. La API REST identifica a los usuarios utilizando autenticación con *token*; las respuestas están disponibles en JSON.

La API no posee restricciones en cuanto al número de peticiones por usuario, sin embargo, se dará respuesta única y exclusivamente a peticiones hechas desde el servidor configurado, es decir, que inicialmente no está previsto ofrecer soporte para aplicaciones externas a *Shader Tool*.

La API fue dividida en 5 recursos principales: usuarios, modelos, texturas, *shaders* y auth. Para acceder a ella se debe incluir el prefijo “api/0.1.0” en la URI del recurso, por ejemplo example.com/api/0.1.0/users. Los parámetros de consulta soportados se indican en la **Tabla 17**.

Atributo	Valores permitidos	Descripción
dummy	{true, false, 0, 1}	Retorna la estructura de la tabla asociada a un recurso.
data_lenght	{min, médium, all}	Específica la cantidad de campos del recurso solicitado se desea obtener.
page	Número >= 1	Permite obtener la página indicada de un recurso.

per_page	Número >= 1	Permite indicar cuantos registros se desean obtener en la página solicitada.

Tabla 17 - Campos válidos para ser usados en la cadena de consulta de una petición.

Existen restricciones en cuanto al uso de los atributos anteriormente descritos y las URIs de un recurso. Los atributos de división de recursos por página y de estructura (*page, per_pega y dummy*) solo pueden ser incluidos en servicios raíz de los recursos, por ejemplo GET: example.com/api/0.1.0/users. Por otra parte, el atributos data_lenght puede ser utilizado en servicios que den acceso a uno o varios recursos de la forma GET:…*/resources* y GET:…*/resources/{id}*. En ningún caso pueden ser utilizados con servicios pertenecientes al grupo *auth*.

4.6.1 Servicio *Modes* y *Textures*

El recurso modelo y textura fueron incorporados en la solución para hacer un manejo dinámico de estos. Cada usuario cuenta con la posibilidad de crear una biblioteca propia de modelos e imágenes que podrán usar en el desarrollo de sus *shaders*. Los recursos están representados como se muestra en la **Tabla 18**.

Model	
Id (int)	Vertices (long text)
Name (String)	Faces (long text)
Format (String)	Uvs (long text)
Size (int)	

Texture	
Id (int)	Size (int)
Name (string)	Key (string)
Format (string)	Path (string)

Tabla 18 - Atributos presentes en el objeto *Model* y *Texture*.

Los URIs asociados a ambos recursos se describen en la **Tabla 19**.

Método HTTP	URI
GET - POST	api/0.1.1/models
GET- PUT - DELETE	api/0.1.1/models/{id}
GET	api/0.1.1/sample-models
GET – POST	api/0.1.1/textures
GET- PUT - DELETE	api/0.1.1/trextures/{id}
GET	api/0.1.1/sample-textures

Tabla 19 - URIs asociados a los recursos *Texture* y *Model*.

El objetivo de los servicios simple-texture y simple-models es acceder de forma directa a los recursos que fueron incluidos en la biblioteca pública de recursos. Esto permite a la solución cliente obtener recursos clasificados como texturas o modelos sin necesidad de proveer información adicional. Ambas biblioteca pueden ser gestionadas y actualizadas de manera independiente a la solución cliente.

4.6.2 Servicio *Shaders*

Los *shader* representa el elemento principal dentro de la solución. Este recurso permite manipular los datos referentes a los programas de *shaders* (*Vertex Shader* y *Fragment Shader*). Cada *Shader* es asociado a un usuario y a un único modelo a través del identificador (*model_id*). De la misma manera, es posible asociar varias imágenes al *shader* desde la biblioteca privada del usuario o pública de la solución. La representación del recurso se muestra en la **Tabla 20**.

Shaders		
Id (int)	Fragment (long text)	Model_id (int)
Name (String)	Uniforms (long text)	User_id (int)
Vertex (long text)		

Tabla 20 - Atributos presentes en el objeto *Shader*.

Los URIs registrados para este recurso se muestran en la **Tabla 21**. El servicio "shader template" provee de forma directa los *shaders* de ejemplo que son mostrados en la solución cliente.

Método HTTP	URI
GET - POST	api/0.1.1/shaders
GET- PUT - DELETE	api/0.1.1/shaders/{id}
GET	api/0.1.1/shaders-templates

Tabla 21 - URIs asociados al recurso *Shader*.

4.6.2 Servicios *Users* y *Auth*

Toda la información que se genera en la solución debe estar asociada a un usuario. A través de un identificador es enlazado un usuario con el resto de los recursos añadidos a su cuenta. Así mismo, la información es usada para permitir el acceso a la solución y poder usar la interfaz de usuario para desarrollar los *shaders*.

A diferencia de los recursos inherentes al funcionamiento de la solución, los servicios *Auth* son definidos en el mismo grupo que el resto, sin embargo, son implementados por el componente de manejo de inicio de sesión. Los campos incluidos en el recurso usuario se muestran en la **Tabla 22**.

Users		
Id (int)	Lastname (string)	image (string)
Name (String)	email (string)	Username (string)

Tabla 22 - Atributos presentes en el objeto User.

Para disponer de una forma flexible y directa de manipular los recursos asociados a un usuario, se definieron servicios especiales que permiten obtener un listado de recursos, texturas o *shaders* de un usuario en particular solo con indicar su identificador. Esto permite que el número de consultas sea reducido, y la cantidad de información que se requiere ordenar y clasificar del lado cliente sea significativamente menor. Los servicios asociados recurso usuario se muestran en la **Tabla 23**.

Método HTTP	URI
GET - POST	api/0.1.1/users
GET- PUT - DELETE	api/0.1.1/ users/{id}
PUT	api/0.1.1/users/{id}/password
PUT	api/0.1.1/users/{id}/image
POST	api/0.1.1/ users/exist
GET	api/0.1.1/users/{id}/models
GET	api/0.1.1/users/{id}/textures
GET	api/0.1.1/users/{id}/shaders
GET – POST - DELETE	api/0.1.1/auth

Tabla 23 - URIs asociados al recurso User.

El último servicio mostrado en la **Tabla 23** son los que permiten crear una sesión para identificar a cada usuario. El método más importante es POST ya que es el que recibe los datos de *username* y *password* indicados por el usuario para luego ser procesados. De ser válidos, el método asociada genera un Token[25] que identifica la sesión.

[25] Firma cifrada que permite a un API identificar al usuario.

4.7 Arquitectura de la solución del lado cliente

La solución fue diseñada con el esquema *single-page* usando el marco de trabajo AngularJS. El framework de JavaScript de código abierto, permitió separar la manipulación DOM[26] de la lógica de la solución siguiendo patrón MVC de ingeniería de software.

Angular JS define una arquitectura formada por ocho módulos principales: aplication root, module, config, route, view, controller, diretctive y factories/services. La interacción entre ellos se muestra en la **Figura 18**.

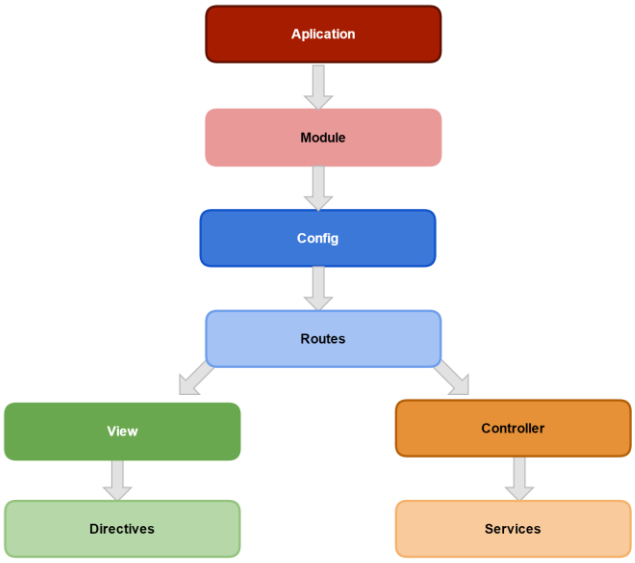


Figura 18 - Módulos de la arquitectura provista por AngularJS.

- ***Aplication***: Representa el elemento raíz de la solución del lado cliente. Angular dispone del atributo *ngApp* para indicar dentro del documento HTML cual elemento se utilizara para iniciar la aplicación.

- ***Module***: Permite incluir en el sistema componentes externos. Estos componentes son de propósito general y ayudan a potenciar las características de la solución con nuevas funcionalidades.

- ***Config***: Permite establecer la configuración de diversos parámetro de inicio y ejecución de la solución. En este módulo se hacen ajustes relacionados con datos de conexión, permisos de acceso, cache, debug, entre otros.

- ***Route***: Se utiliza para enlazar las URLs a los controladores y las vistas. Permite definir rutas a través de la API del objeto $uirouteProvider.

[26] Es la estructura de objetos que genera el navegador cuando se carga un documento

- ***Controller***: Se utilizar como funciones constructoras JavaScript que permiten aumentar el alcance del framework Angular. Los controladores se enlazan al documento DOM y se crea un nuevo ámbito cuanto este es invocado.

- ***View***: Constituyen todos los archivos HTML que componen a la solución. El documento principal fue nombrado Index.html y empleando la técnica AJAX, se inyecta las vistas (plantillas) en el nodo configurado con el atributo *ngView*.

- ***Directives***: Permite declarar componentes Web. Encapsulan comportamientos y plantillas que pueden ser usados en las vistas mediante un atributo o nombre clave.

- ***Services***: Se puede describir como una clase que contienen la representación de un objeto o proceso. Los servicios son usados en los controladores para dividir los procesos en módulos y así optimizar la reutilización de código.

La solución se conforma por aproximadamente 10 vistas. Durante el desarrollo se implementaron varios componentes Web, factorías y servicios relevantes para el funcionamiento de la solución. Entre los módulos más importantes se pueden mencionar: *jQuery*, *Bootstrap*, *UI-Layout*, *ngCropper* y *uiAce*.

Por otra parte, los servicios conforman gran parte de la solución del lado cliente, ya que modelan la mayoría de los objetos utilizados para gestionar toda la información y eventos generados por el usuario.

La solución cliente está formada por tres paquetes principales que hacen uso de los diversos módulos mencionados para ordenar y estructurar las funcionalidades. *Graphics* por su parte, ordena los diferentes objetos involucrados en el proceso de carga, procesamiento y despliegue de un programa de *shader*. *Directives* contiene todos los componentes Web desarrollados. Y el paquete *Filters* define pequeños procesos que permiten transformar objetos o cadenas de caracteres para ser mostradas en la vista con un patrón en particular.

4.8 Paquete *Graphics*

En el proceso de despliegue de un *Shader* son varios los procesos involucrados y las etapas que deben cumplirse para finalmente desplegar un modelo en el cual se aplique el programa GLSL. Este paquete define los diferentes objetos que intervienen en dicho proceso, y validan que cada etapa se cumpla de forma exitosa.

4.8.1 Servicio *Model*

Representa una instancia del recurso modelo y provee funciones que permiten tener la información sincronizada entre el cliente y el servidor. El método *load permite* solicitar al API REST un modelo mediante su identificador. Esta función se encarga de establecer la conexión y de configurar sus atributos, de tal forma que al procesar la respuesta, el modelo tridimensional esté disponible para su uso.

También es posible crear una instancia de la clase a través del método *init*. A diferencia del método anterior, este recibe como parámetro los datos de modelo que se requieren usar, por tanto no es necesario realizar peticiones. Finalmente provee la posibilidad de guardar el estado actual del modelo directamente en la base de datos del API para su posterior uso. El método *save* recibe como único parámetro el id del usuario propietaria, ya que este campo es requerido en el proceso servidor.

El formato de modelo admitido en la solución es JSON, sin embargo, es posible incluir en versiones posteriores formatos adicionales como colada y OBJ. Para disponer de un mecanismo estándar de almacenamiento y carga de modelos, la representación del modelo almacenado es la provista por el framework ThreeJS.

Para guardar un modelo (Three.Mesh) se hace una representación del objeto como si se tratara de una cadena de caracteres y es finalmente esta representación la que se usa en los datos enviados junto a la petición POST. Una vez almacenado, esta información puede ser solicitada y representada nuevamente como un objeto JavaSctipt valido.

El servicio encargado de realizar ambos procesos es *ModelFileUtil*, el cual posee varios métodos que ayudan a obtener diversas representaciones del mismo objeto a través de una fuente de dato. En la **Tabla 24** se muestran todos los métodos que ofrece.

Método	Descripción
geometryToString(geometry)	El objeto *Mesh* de la biblioteca *Three*, entre otros atributos define geometry. Este representa toda la información relacionada a la geometría del modelo. Esta estructura de dato es recibida como parámetro y es trasformada en una cadena de caracteres. Este proceso se conoce como con el nombre *Stringify*. El método retorna un objeto con la representación de las caras, vértices y uvs del modelo.
stringToGeometry(vertices, faces, uvs)	Toma tres cadenas de caracteres (*vertices*, *faces* y *uvs*) e intenta obtener su representación dentro de un objeto geometry. Adicionalmente, el método procesa las normales por vértice y caras de la geometría obtenida.

parseJson(file, callback)	Para proveer la carga de modelos al usuarios, la solución permite seleccionar un archivo en formato JSON que represente a un modelo 3D. Este método procesa el archivo y retorna un objeto *Mesh* que representa a dicho modelo.

Tabla 24 - Métodos presentes en el servicio *ModelFileUtil*.

4.8.2 Módulo *Shader*

Este módulo es el encargado de gestionar todos los aspectos relacionados a la carga, creación, compilación y respaldo de los *shaders*. El proceso inicia con el servicio *Shader* que representa al recurso, en la **Figura 19** puede ver las propiedades y métodos con los que cuenta.

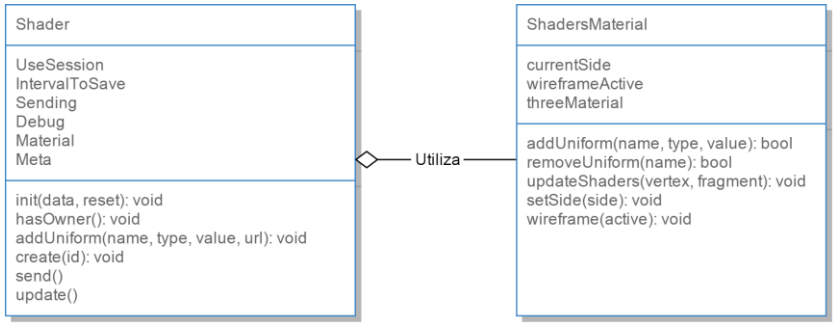


Figura 19 - Diagrama de clases del módulo *Shader*.

Los atributos y métodos de mayor relevancia se describen en la **Tabla 25**.

Atributo/Método	Descripción
BOOL: _sending	Indica el momento en que el *shader* está siendo sincronizado del cliente hacia el servidor.
BOOL: _debug	Despliega cierta información de control en la consola JS que puede ser utilizada para realizar pruebas y validaciones a los procesos del servicio.
INT: intervalToSave	Indica el intervalo en milisegundos en que se debe realizar la sincronización.
OBJECT: meta	Contiene el conjunto de atributos que almacena cada campo del recurso *Shader*. A través de él, se puede acceder al código del programa de fragmento, programa de vértice, atributos, etc.
Init(meta, reset)	Permite iniciar la clase con la representación de un recurso *shader*.
hasOwner()	Retorna verdadero si el *shader* está asociado a un usuario. Falso en caso contrario.

addUniform(name, type, value, url)	Permite registrar variables uniformes al *shader*. Los parámetros solicitados son el nombre de la variable, el tipo de dato, su valor, y en el caso que la variable sea de tipo textura se debe indicar el URL donde se encuentra la imagen.
removeUniform(name)	Elimina una variable uniforme del *shader*.
create(id)	Este método permite crear un nuevo recurso *shader* del lado del servidor. Para ser almacenado, se debe indicar como parámetro el id del usuario propietario.
send()	Este método envía el estado actual del recurso al servidor para ser actualizado. Este método se invoca de manera automática cada cierta cantidad de milisegundos (***intervalToSave***).
update()	Este método permite hacer la actualización del programa de *shader* en la tarjeta gráfica del computador.

Tabla 25 - Descripción de atributos y propiedades del servicio *Shader*.

En el proceso de actualización del *shader* intervienen dos servicios que están involucrados en la compilación y reporte de error. *ShaderCompiler* dos métodos para compilar cada programa: *compileVertex* y *compileFragment*. Cada método retorna un objeto con el estado de la compilación y de ser necesario la lista de errores obtenidos. Este objeto es recibido por el servicio *PrepareLog*, lo procesa y presenta con una estructura más apropiada para ser manipulada desde el controlador y vista.

4.8.3 Servicio *Camera*

La cámara con perspectiva es la utilizada en la solución de manera predeterminada. Todas las funcionalidades para su uso fueron implementadas en el servicio *PerspectiveCamera*. Esta provee un manejo abstracto del objeto cámara proporcionado por ThreeJS, y define métodos directos para cambiar sus propiedades.

El servicio recibe como parámetros la crear su instancia el angulo de visión (*field of view*), la relación del aspecto (*aspectRatio*) y la profundidad del plano frontal y trasero (*near - far*). De la misma manera, permite actualizar en cualquier momento de la ejecución estos valores.

Para proveer la interacción entre el usuario y el objeto cámara se utilizó el objeto *Three. TrackballControls.* Este define las interfaces necesarias para rotal la cámara alrededor de un punto, cambiar la posición de profundidad (zoom) o desplazarla a través de la escena.

4.9 Paquete *Directives*

Las directivas tienen el aspecto de elementos o atributos en el código HTML, sin embargo, estas son interpretadas por angular al generar las vistas, lo que permite modificar el DOM o añadir nuevos comportamientos diseñados de manera independiente.

Se diseñador componentes de diversas índoles, sin embargo, las directivas principales implementadas son las encargadas de proveer acceso al editor de texto, pre visualización de recursos al cargar y mensajes de reporte.

4.9.1 Directiva aceEditor

Ace es el nombre de la biblioteca usada para incluir el editor de texto en la solución. Este editor es de código embebido escrito en JavaScript. Coincide con las características y el rendimiento de los editores nativos como Sublime, Vim y TextMate. Puede ser fácilmente integrado en cualquier página Web y aplicaciones JavaScript.

Para su uso se creó la directiva aceEditor, y conceptualmente es utilizada como un *wrapper*[27] que permite configurar la instancia del editor de texto. Esta directiva declara 3 atributos los cuales son: *source* representa la fuente de datos que será mostrado en el editor, *onChange* la función *callback* que debe ser invocada al ocurrir un cambio de estado y *name* que se usa como identificador el elemento HTML.

4.9.2 Directivas de pre visualización de recursos

La solución a través de dos interfaces permite cargar y pre visualizar archivos en formato JSON, JPEG y PNG con el objetivo de incluirlos en la biblioteca de recursos de modelos y texturas. Ambas interfaces involucran los procesos de selección, despliegue y aprobación. Con el objetivo de contar con una interfaz programática que permitiera la gestión de estos archivos, se diseñaron componentes Web que permiten seleccionar y procesar dichos recursos.

En el ámbito de las imágenes, la directiva encargada es *textureWizard*. Este componente encapsula todos los procesos involucrados en agregar una nueva imagen a la bilbioteca. Inicialmente ofrece un área que permite acceder el explorador de archivo del dispositivo (PC o móvil) para seleccionar el archivo de imagen. Una vez seleccionada, se procesa y se muestra la pre-visualización, permitiendo descartarla o guardarla. Le código que permite habilitar este componente se muestra en el **Código 8**.

```
<texture-wizard
 on-success-save="callback"
 on-fail-save="callback"
 on-saving="callback">
</texture-wizard>
```

[27] Programa o *script* que establece las bases y hace posible el funcionamiento de otro programa más importante. También conocido como programa o *script* de delegación.

Código 8 - Etiqueta HTML que define el componente *texture-wizard*.

El proceso y funcionamiento para la gestión de los archivos JSON es similar al explicado anteriormente. La directiva fue nombrada *modelWizard* y las diferencias más notables son la configuración dada al selector de archivo, y el pre-visualizador. Al tratarse de modelos es necesario componer una escena 3D simple y ella desplegar el modelo seleccionado de resultar valido.

Es importante indicar que tanto la directiva *textureWizard* como *modelWizard* pueden ser incluidas en cualquier parte de la solución y contar con las funcionalidades que ofrece de manera inmediata.

4.9.3 Directiva MessageBox

La directica caja de mensaje (*MessageBox*) es utilizada en toda la solución del lado del cliente para desplegar mensajes informativos o de reporte de algún proceso u operación solicitada por el usuario. Por la característica dinámica de los mensajes, la directiva se apoya en un servicio llamado AlertMessage que es el encargado de registrar el mensaje que posteriormente la directiva lo despliegue.

AlertMessage provee cinco métodos para hacer el registro diversos tipos de mensaje: default, info, warning, success y error. Cada uno de los mensje establece una configuración por defecto con elación a color, titulo e icono a usar, sin mebargo, esta puede ser ajustada sin inconvenientes. El proceso para desplegar un mensaje informativo es el siguiente:

1. Se incluye la directiva *messageBox* en el documento HTML.
   <*messageBox*></*messageBox* >
2. Desde cualquier controlador se hace uso del método *info(data)* provisto por el servicio *AlertMessage*.
3. El método recibe un objeto llamado "*data*" como parámetro, que contiene: título, mensaje, tiempo de duración e icono. Estos datos son usados en la instancia de la directiva.

Finalmente, el mensaje es desplegado en la pantalla durante la cantidad de segundos configurados. De ser necesario el usuario puede removerlo antes de este tiempo haciendo clic en la x dispuesta en cada mensaje.

4.10 Módulo *filters*

Los filtros son una herramienta que permiten estandarizar procesos simples y recurrentes dentro de la manipulación de los datos al ser desplegados en la vista. Permite que dados unos datos de entrada, mediante parámetros, obtengamos una salida depurada de acuerdo a las necesidades.

La utilidad de los filtro en amplia y pueden ser usados para tareas simples como formatear una fecha y limitar el número máximo de resultados, o para procesos más complejos como procesar grandes estructuras de datos para ordenarlos o realizar búsquedas. En la **Tabla 26** se muestran los filtros implementados.

Filtro	Parámetros	Descripción
reverseArray	Array	Invierte el orden de un arreglo. Ejemplo: [1, 5, 9] » [9, 5, 1]
unique	Array	Permite eliminar los valores duplicados de un arreglo. Ejemplo: [1, 5, 9, 1] » [1, 5, 9, 1]
hexaToRGB	Color: String	Trasnforma un color hexadecimal a su representación en formato RGB. Ejemplo: #FF0000 » {r:255, g:0, b:0}
rgbToHexa	{b:int, g:int, b:int}	Transforma un color RGB a su representación en formato hexadecimal. Ejemplo: {r:255, g:0, b:0} » #FF0000
glType	Type: String	Permite mostrar los tipos de datos manejados por los *shaders* de manera más simple al usuario. El filtro toma la forma de representación establecida por ThreeJS y la adapta a una más descriptiva. Ejemplo: v2 » Vector 2
byteTo	Size: String/int, measure: String	Dado un valor el cual representa una cierta cantidad en bytes, es transformada de acuerdo a la escala (*measure*) solicitada. Ejemplo: 8192, KB » 8KB

Tabla 26 - Lista de Filtros implementados.

Adicionalmente, existen numerosos filtros para el manejo de cadenas de caracteres. Algunas de ellas son: *capitalize*, *decapitalize*, *countSubstr*, *escapeHTML*, entre otras.

4.11 Solución *Shader Tool*

La solución interactiva permite mostrar las funcionalidades implementadas y anteriormente explicadas. Su propósito es permitir al usuario interactuar con el sistema de manera gráfica haciendo uso de la entrada estándar de teclado y ratón.

La solución puede ser definida como un sistema integrado de desarrollo que permite implementar programas de *shaders* para el lenguaje GLSL. Esta dispone de diferentes secciones y componentes que permiten lleva a cabo esta labor.

La interfaz gráfica está dividida en cinco grandes secciones. Cada una de estas secciones está agrupada en sub-elementos según sus características, funcionalidades y similitudes. Por ejemplo, todos los comandos para la programación de los programas de *shaders* se encuentran agrupados en

la parte izquierda de la barra de menú. Como puede verse en la **Figura 20**, las cinco secciones principales son las siguientes:

- **Barra de menús**: corresponde a la fuente de comandos más común. En la barra de menús se encuentran todos los métodos y funcionalidades de la solución. Está dividida en tres secciones bien marcadas: opciones de editor, opciones de despliegue y perfil de usuario.

- **Panel de Editor**: Define el área en la cual se escribe el código de los programas de *shaders*. Cada programa (de vértice y fragmento) disponen de este espacio, sin embargo, solo uno podrá estar activo en un momento dado.

- **Panel de vista**: Permite percibir el contenido de la escena, en la cual se aplica el resultado tras la compilación de ambos programas de *shaders*. Junto al panel de editor, puede ser ajustado su tamaño a gusto del usuario.

- **Área de resultado**: Muestra el resultado obtenido tras la compilación del *shader*. Está ubicada en la parte inferior del panel de vista, y agrupa en dos pestañas los mensajes para cada programa.

- **Área de notificaciones**: Despliega las notificaciones generadas por procesos iniciados por el usuario. Está ubicada en la parte superior derecha del panel de vista.

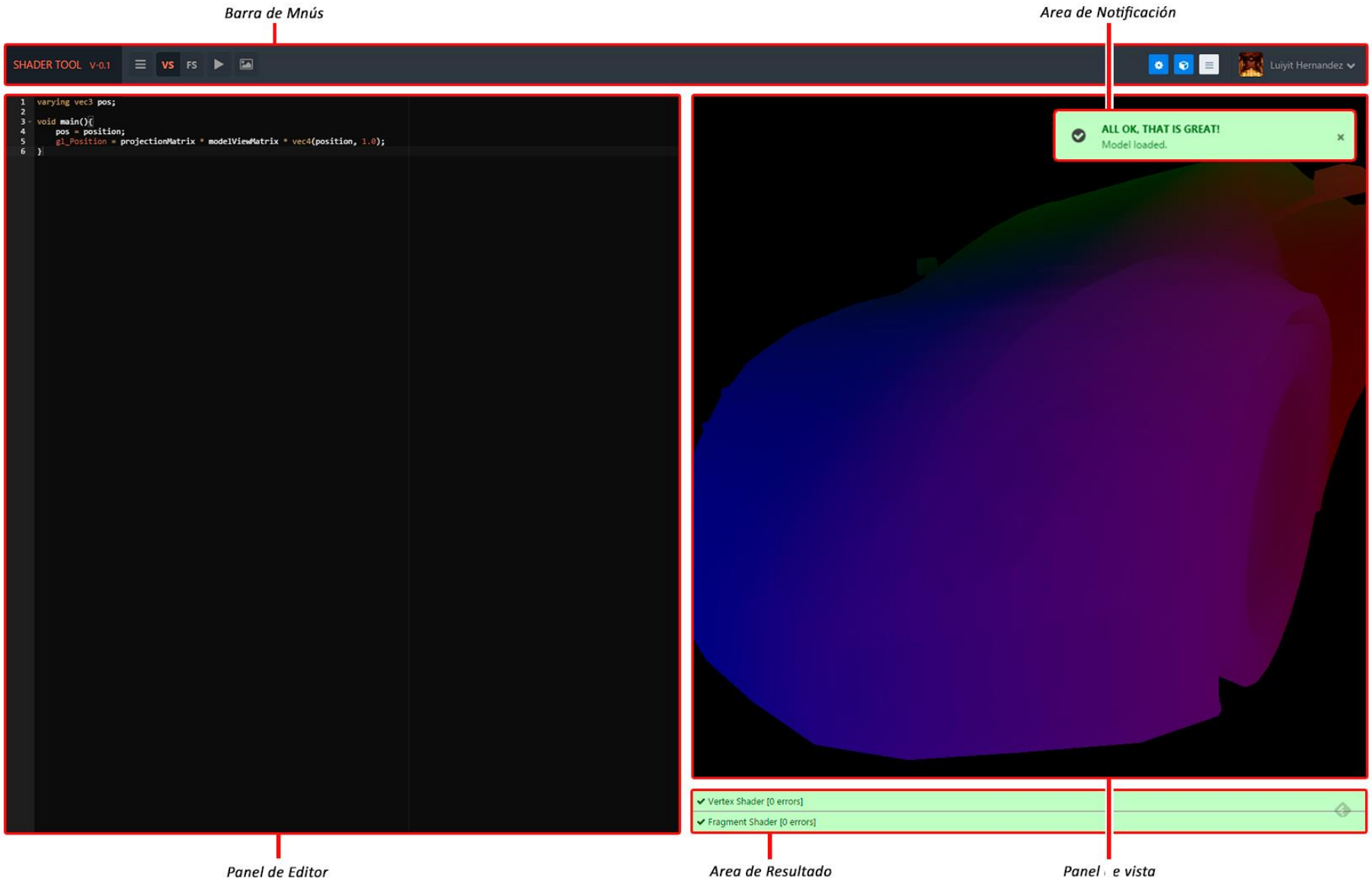


Figura 20 - Secciones de la página editor (solución del lado cliente)

4.11.1 Barra de menúes

Los menús desplegables y botones ubicados en la parte superior de la ventana principal incluyen muchas de las funcionalidades disponibles en la solución *Shader Tool*. Muchos de los comandos disponibles se encuentran en menúes contextuales en los diversos paneles. Para ejecutar un comando ubicado en alguno de los menús o botón, puede hacerse clic sobre él utilizando el cursor del ratón.

En la **Figura 21** puede apreciarse la barra de menús principal del lado izquierdo la cual contiene las siguientes opciones: Archivo (File), Cambio de programa (Swap), Ejecutar (Run) y biblioteca de Texturas (Textures Library). Las opciones solo se encuentran disponibles al estar dentro del área del editor de *shaders*, en otras vistas las opciones no se muestran al usuario.

Figura 21 - Barra de menús izquierda.

No todos los comandos de los menús están siempre disponibles. Si un comando no está disponible en un momento dado, éste se mostrará con un color más claro, y al colocar el cursor del mouse sobre el aparece un indicador en rojo. Dependiendo del estado del sistema los diferentes comandos se activarán y desactivarán de los menús. Por ejemplo, al entrar al editor los comandos "*Shader Details*" del menú File, se encuentran desactivados, y son habilitados solo después de que un *shader* haya sido cargado en el sistema.

Del lado derecho de la barra de menús principal mostrado en la **Figura 22**, se encuentran las siguientes opciones: Configuración (*Settings*), Seleccionar modelo (*Select Model*), herramientas de despliegue (*Render Tools*) y perfil (*Profile*).

Figura 22 - Barra de menús derecha.

4.11.2 Barra de menúes principal a detalle

A continuación se lista el contenido de cada menú y se detalla la funcionalidad de cada uno de los comandos.

- **El menú File**: En este menú se encuentran los comandos para visualizar el detalle del *shader* cargado (*Shader Details*), crear un nuevo shader (*New Shader*), cargar un shader de la biblioteca del usuario activo (*Load Shader*), descargar el shader actual (*Download current Shader*), pantalla completa (*Fullscreen*) y ayuda (*Help*).

- ***Togle Shaders***: Cuenta con dos botones los cuales permiten cambiar entre ambos programas de *shader*. El programa activo es resaltado con un color naranja.

- **Botón Ejecutar *Shader*:** Permite compilar el código de los programas. En el tiempo que tarde el proceso, se mostrara un indicador en el botón para que el usuario tenga conocimiento del estado de la operación.

- **Biblioteca de Texturas:** Da acceso al panel de gestión de texturas del usuario, en donde se mostraran las imágenes que tiene actualmente agregadas, las imágenes que ofrece la solución de forma estándar a todos los usuarios y un área donde que permite cargar nuevas imágenes.

- **Configuraciones:** Permite configurar opciones de despliegue asociadas a la cámara y modelo.

- **Biblioteca de modelos:** Muestra el panel de modelos disponibles para usar. También permite la carga de nuevos modelos 3D. A diferencia de la biblioteca de texturas, solo un modelo puede ser asociado al *shader* actual.

- **Herramientas**: Contiene los comandos que permiten descargar la imagen del área de despliegue (Print Canvas), reiniciar a la configuración escala, posición y rotación del modelo y detener o activar la rotación automática de la escena.

- **Perfil**: Se encuentran los comandos de edición de perfil (edit Profile) y salir (Logout).

4.11.3 Panel Editor de texto

El panel del editor permite a los usuarios escribir el código que componen su *shader*. El editor dispone de tres elementos de interfaz mostrados en la **Figura 23**, que ayudan a la interacción: Número de línea, colapso de reglones de código y menú contextual de edición.

Para colapsar una región de código se debe hacer clic con el cursor del mouse en la flecha indicada en el **Figura 23**, de la misma manera se puede abrir nuevamente la región. Con respecto al menú contextual ofrece opciones de deshacer y rehacer el estado del editor, seleccionar todo el código y los comandos estándar copiar, pegar y cortar.

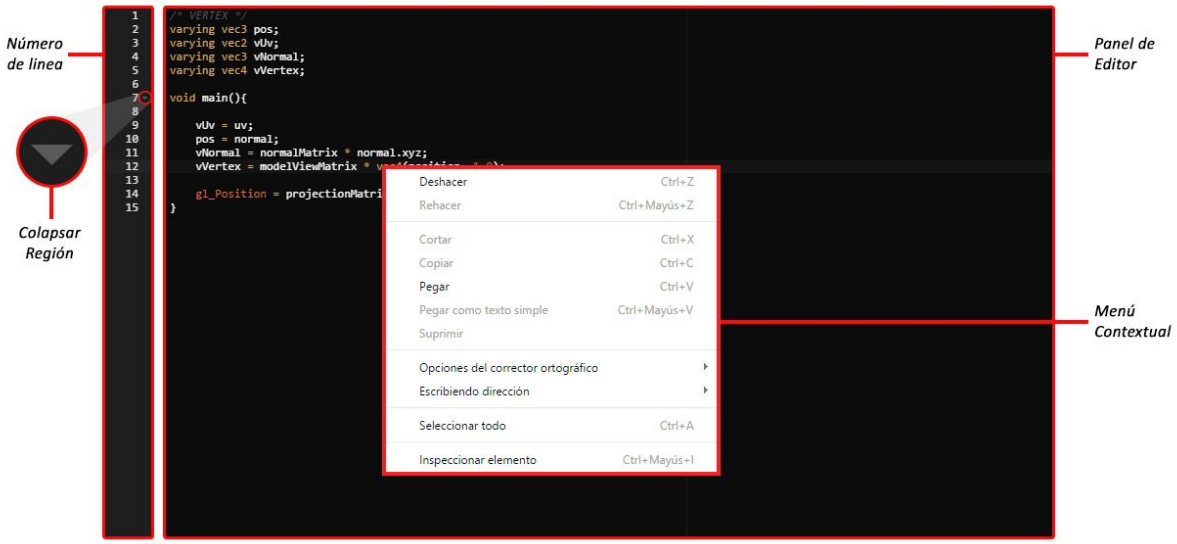


Figura 23 - Panel de editor.

4.11.4 Área de despliegue (*Viewport*)

Un *viewport* puede definirse como una ventana en donde se despliega la escena vista desde cierta perspectiva. Esta ventana posee numerosas opciones que le permiten al usuario diferentes configuraciones al momento de visualizar la escena.

La primera opción de configuración es la posibilidad de ver la escena con una vista perspectiva u ortográfica (también llamada ortogonal).

La vista ortográfica se caracteriza porque los elementos de la escena son visualizados de forma directa desde un lado o “cara”. Esto revela la vista de la escena desde un único plano. Adicionalmente, cuando se observa directamente la cara frontal de un objeto desplegado se distinguen el ancho y altura del mismo en dos dimensiones, pero no la tercera dimensión, la profundidad. Cada vista ortográfica proporciona dos de las tres dimensiones principales.

En contraparte, la visión perspectiva simula la manera como nuestros ojos realmente funcionan. Todos los puntos convergen a una única posición, lo cual recrea la profundidad en la escena y produce el efecto de reducción de tamaño en los objetos. En la **Figura 24** se pueden apreciar las diferencias entre estas dos maneras de visualización.

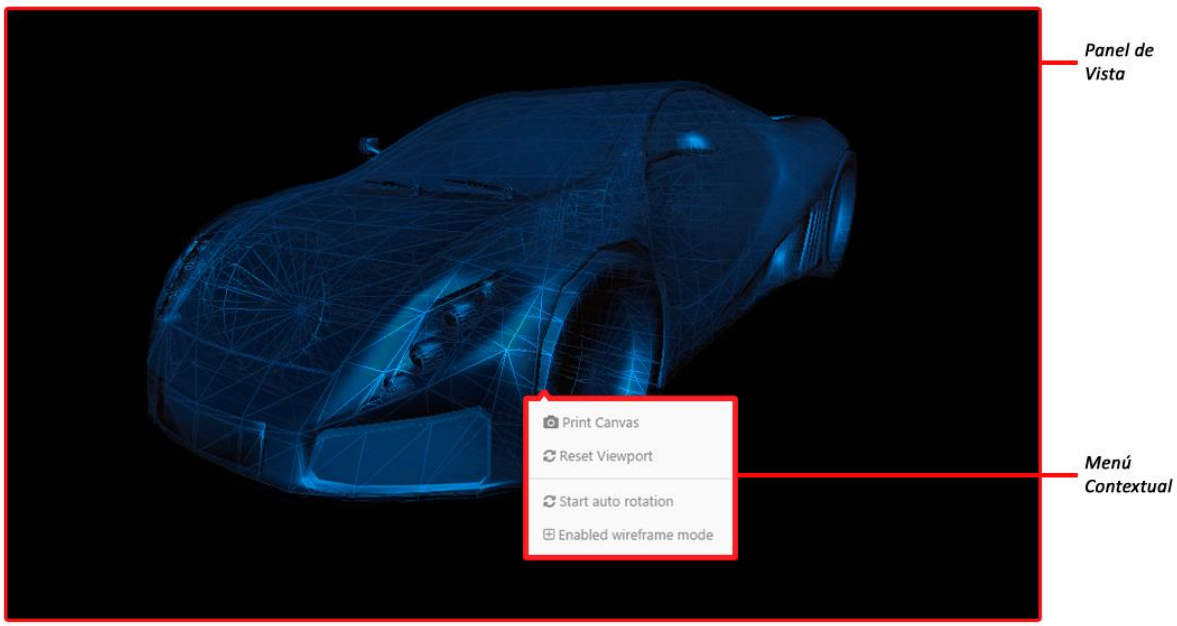


Figura 24 - Panel de vista (*Viewport*).

Otro elemento de configuración del *viewport* son los atributos con los cuales se despliega el objeto en la escena. Los parámetros que pueden ser adaptados por el usuario de acuerdo a sus necesidades son:

- Lado de las caras (*sides*) que se deben desplegar. Puede seleccionarse entre Fontal, trasera o ambas.
- Activar el uso de buffer de profundidad (*zBuffer*). Este le permite a la tarjeta gráfica reproducir correctamente la percepción de profundidad normal: los objetos cercanos ocultan a los más lejanos.
- Desplegar la estructura del objeto como alambre (*Wireframe*).

La cámara también puede ser adaptada y pueden ser ajustados valores como posición del plano de corte cercano y lejano, y el ángulo de visión (*Field Of View*) en el caso que se utilice la cámara con perspectiva. Es importante destacar que estas opciones de configuración son accesibles desde la barra de menúes principal, y a través del menú contextual que puede visualizarse haciendo clic con el botón derecho del mouse en cualquier área del *viewport*.

11.4.5 Área de resultado

Esta sección de la solución es la que le permite al usuario verificar los posibles errores que contiene el código de sus programas de *shaders*. El área de resultado muestra de manera independiente los resultados para el programa de vértice y de fragmento. De la misma manera, mostrara indicadores visuales para hacer evidente los errores presentes en el código.

Tras la compilación exitosa de ambos programas, se mostraran dos barras de estado en color verde como se muestra en la **Figura 25** que indican que no se encontraron errores en el código.

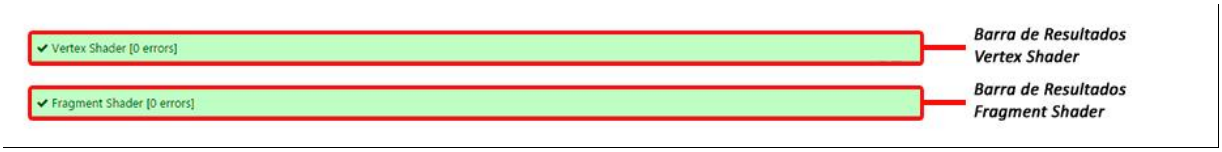


Figura 25 - Área de resultado de compilación de *Shaders*.

En contraparte, de existir algún error, estos serán clasificados en ambas barras y se mostrara la lista de errores en color rojo. Entre los datos mostrados asociados a un error en particular se encuentras: la línea en donde se detectó el error, la variable o palabra involucrada y la descripción del error. En la **Figura 26** se detallan cada uno de estos elementos. De resultar solo un programa con errores detectados, en su barra correspondiente se mostraran los errores, mientras que en la otra se indicara que la compilación fue exitosa. Para poder ser procesado el *shader*, ambos programas deben compilar de forma exitosa.

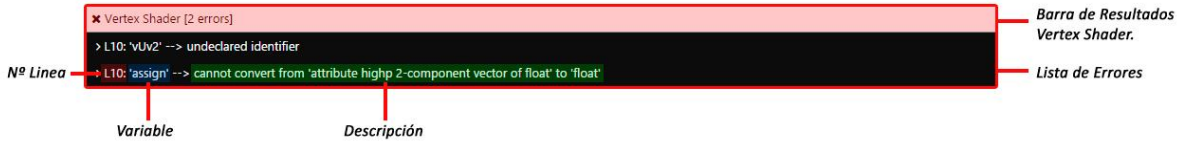


Figura 26 - Elementos presentes en la barra de resultado al existir un error de compilación.

4.11.6 Área de notificaciones

Las notificaciones son la forma más directa de trasmitir un mensaje al usuario. Las notificaciones que utiliza la solución son mostradas para informar el estado de alguna operación. Son muchas las notificaciones que son mostradas al utilizar las funciones de la solución, en el **Figura 27** se muestran los diversos mensajes de alertas programados.

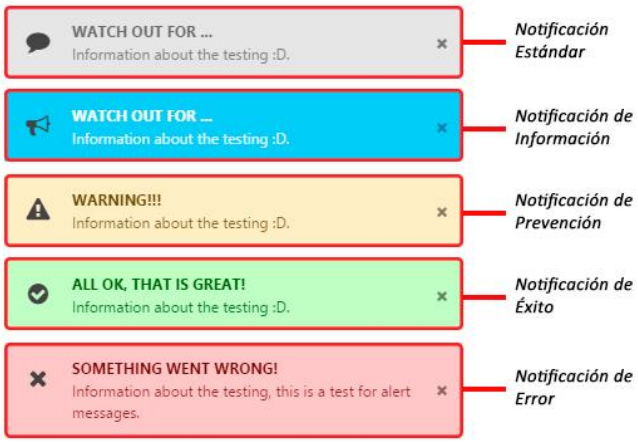


Figura 27 - Tipos de alertas de notificación.

Las notificaciones son mostradas una debajo de la otra en el lado superior derecho de la ventana del navegador. El tiempo visible programado es de 8 segundos, sin embargo, el usuario tiene la opción de cerrar el cuadro del mensaje en cualquier instante.

4.11.7 Modales funcionales

La solución *Shader Tool* cuenta con un gran número de opciones y herramientas de configuración asociados al proceso de programación de los *shaders*. Teniendo en cuenta que el espacio más relevante de la ventana del navegador fue apropiadamente dispuesto para los dos paneles principales, se empleó un sistema de modales para agrupar funcionalidades particulares. De esta manera las diferentes funciones de la solución son accedidas a demanda.

Cada modal se superpone a los paneles principales. Esto permite Ordenar la información y controles en un espacio independiente sin alterar la disposición de la interfaz del usuario. Las funciones más relevantes que usan el sistema de modal son: crear *Shader*, cargar *Shader,* biblioteca de texturas, biblioteca de modelos y perfil de usuario.

4.11.7.1 Modal Crear *Shader*

El modal permite crear un nuevo recurso *shader* en el lado servidor. Al iniciarse el editor completamente se muestra un *shader* por defecto. El objetivo es dar una plantilla iniciar de trabajo al programador, pero es importante tener en cuenta que esa información no está guardada. Si el usuario edita la plantilla inicial, ese código estará disponible mientras la sesión esta activa. Una vez el usuario cierre la sesión de forma explícita o implícita se perderá todo el progreso.

Para respaldar el progreso y recuperar el código en sesiones futuras se debe crear un shader e indicar los parámetros solicitados en el modal. Esta operación puede llevarse a cabo haciendo clic en el menú archivo (file) de la barra de menús principal y seleccionar nuevo shader (*New Shader*). En la **Figura 28** se muestra el modal y los atributos solicitados.

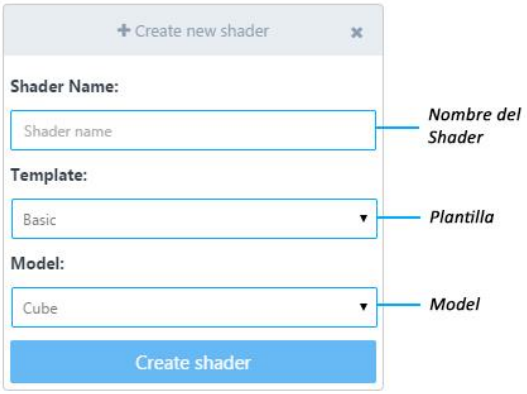


Figura 28 - Modal para crear *Shaders*.

- **Nombre (*Name*)**: Identificará al *shader* en la biblioteca. El usuario es libre de colocar cualquier nombre, incluso es permitido colocar el mismo nombre a dos o más *shaders*.

- **Plantilla (*Template*)**: La solución pone a disposición de los programadores 5 plantillas o ejemplos de *shaders* que pueden ser usados para inicializar el nuevo *shader*. Esto permite tener una estructura inicial. Adicionalmente, e posible utilizar como plantilla el contenido actual del editor de texto. De esta manera los usuarios pueden desarrollar sus *shaders* y guardarlos al finalizar.

- **Modelo (*model*):** Permite seleccionar el modelo que será asociado al *shader*. Los modelos son provistos por la biblioteca de la solución. El modelo puede ser cambiado en cualquier momento, e incluso podrá seleccionarse uno de la biblioteca del usuario.

Una vez creado el *shader* (luego de pulsar en el botón "*Créate Shader"*), el modal se oculta y la plantilla selecciona es cargada en el editor de texto. Des mismo modo se inicia la descarga del modelo seleccionado para ser desplegado en el área del *viewport*. Hay que tener en cuenta que al momento de cargar un *shader* si este no puede ser compilado no se mostrara el modelo, este comportamiento se debe a que la solución no cuenta con un estado anterior valido del *shader* que pueda ser compilado y mostrado.

Toda la configuración del *shader* es guardada de forma automática en *background* sin afectar el flujo de trabajo del usuario.

4.11.7.2 Modal Selector de *Shader*

Los *shaders* implementados y guardados por los usuarios pueden ser accedidos desde el modela de selección de *shaders*. El modal permite visualizar la lista de *shaders* propias de un usuario. Al iniciar la carga se muestra un pequeño recuadro que indica al usuario que la operación de carga se está llevando a cabo (ver **Figura 29**). Al terminar, se muestra la lista antes mencionada, o el acceso directo para crear uno shader nuevo. Esta última, en el caso que el usuario no tenga ninguno guardado.

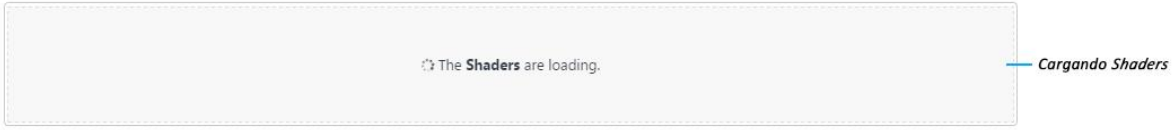


Figura 29 - Indicador de carga de *Shader* guardados.

La lista de *shader* es dividida en páginas formando grupos de cinco, y el *shader* actual es mostrado con un icono en forma de "v" en la columna de opciones de la tabla. Para seleccionar un *shader*, se debe hacer clic sobre el botón de acción azul al final de cada fila. En la **Figura 30** se muestran todos los elementos involucrados en el modal.

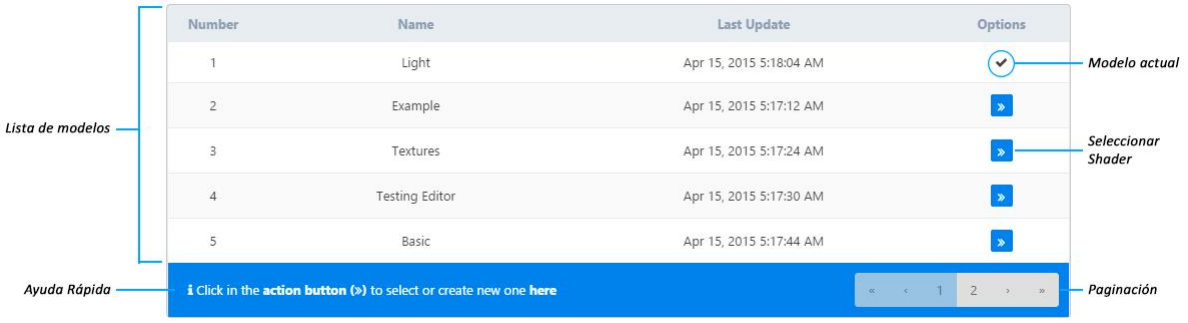


Figura 30 - Lista de *Shaders* asociados a un usuario.

4.11.7.2 Modal Selector de texturas

Para el manejo de texturas se muestra un sistema de pestañas que dan acceso a funciones asociadas. Las pestañas disponibles permiten visualizar y utilizar la biblioteca del usuario, la biblioteca estándar que ofrece la solución, y finalmente las herramientas necesarias para agregar imágenes propias a la biblioteca.

Las dos primeras pestañas muestran un recuadro de carga cuando se están ejecutando operaciones de carga. Cuando esta termina, se muestra la lista correspondiente de texturas. Cada listado muestra los datos de cada textura como su nombre, palabra clave usada al ser usada como variable uniforme dentro del *shader*, si está siendo utilizada en el *shader* actual, entre otros. En la **Figura 31** se muestran las texturas estándar que está disponible para todos los usuarios.

Para seleccionarla o desmarcarla se debe hacer uso del elemento *checkbox* al final de cada fila. Al hacer clic, el estado del *shader* cambia automáticamente y la textura estará disponible o no de manera inmediata.

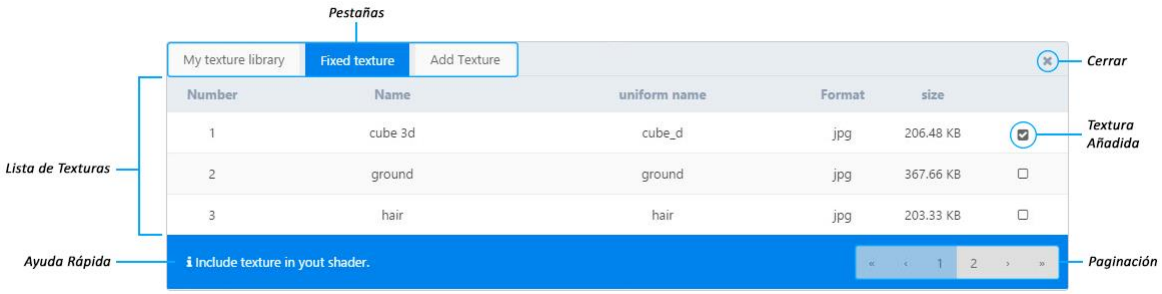


Figura 31 - Lista de texturas provistas por la solución del lado cliente.

El sistema de carga de imágenes mostrado en la **Figura 32** dispone de dos áreas claramente demarcadas. El área a la izquierda permite arrastrar y soltar las imágenes directamente en esa región. Al hacer eso, se realizan las validaciones correspondientes y de ser el caso, se procesa la imagen. Adicionalmente puede hacerse clic en esta región y se mostrara el selector de archivo nativo del dispositivo.

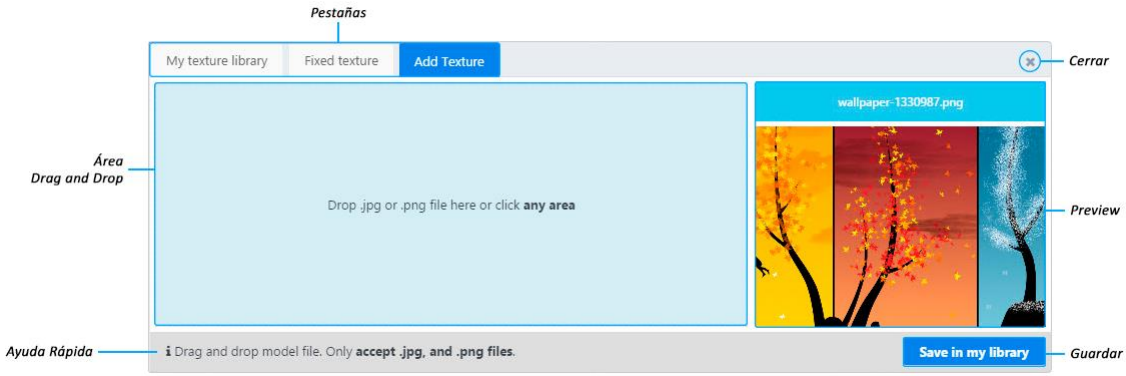


Figura 32 - Área para añadir texturas.

Una vez procesada la imagen se muestra en el área derecha una pre-visualización. Esta región permite manipular la imagen y hacer zoom o desplazarla haciendo uso de los botones del mouse. Junto a la pre-visualización se habilita el botón (*Save in my library*), el cual permite guardar la imagen como parte de la biblioteca de texturas del usuario.

El tiempo de este proceso dependerá directamente del peso en bytes de la imagen y de la velocidad de carga que disponga el usuario en el equipo. Durante el proceso, el modal no puede ser cerrado.

Inmediatamente al terminar la operación y obtener una respuesta afirmativa del servidor, se muestra la notificación correspondiente. También de forma automática se muestra la pestaña de la biblioteca del usuario y resalta el nuevo recurso incluido. En la **Figura 33** se muestra la lista tras añadir una nueva imagen.

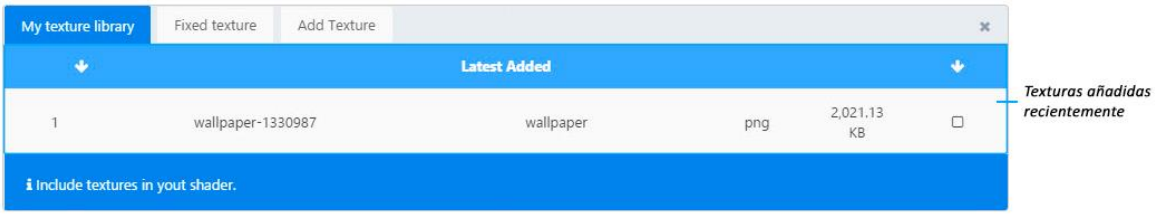


Figura 33 - Lista de texturas añadidas recientemente (acceso directo).

Esta funcionalidad es particularmente útil cuando la biblioteca del usuario es extensa. De esta manera el recurso agregado recientemente se mostrara en el listado principal y no habrá necesidad de navegar hasta la última página de la lista.

4.11.7.3 Modal Selector de Modelos 3D

Al igual que en el modal anterior, al desplegarse el modal *Model Selector* se muestran tres pestañas, dos de las cuales dan acceso a las listas de modelos y la última a la herramienta de carga de

modelos nuevos. La **Figura 34** muestra los modelos disponibles en la biblioteca estándar, entre los atributos desplegados se encuentra: nombre (*name*), formato (Format) y peso del modelo en el momento que fue cargado expresado en *Kilobytes* (*Size*).

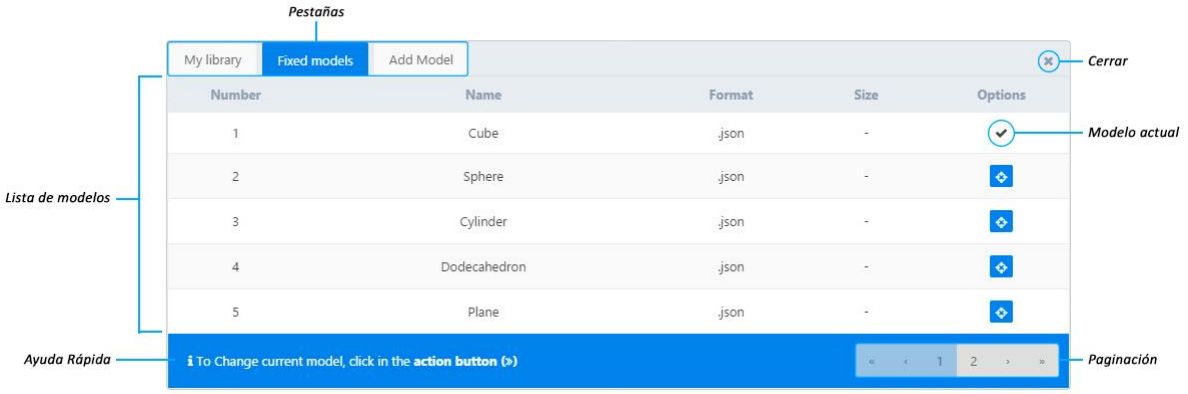


Figura 34 - Modal que da acceso a la lista de modelos.

Haciendo clic con el cursor del mouse en el botón ubicado al final de cada fila, es posible seleccionar el modelo. En este caso, el modal se oculta automáticamente y se mostrará un indicador de carga en el *viewport* como muestra la **Figura 35**.

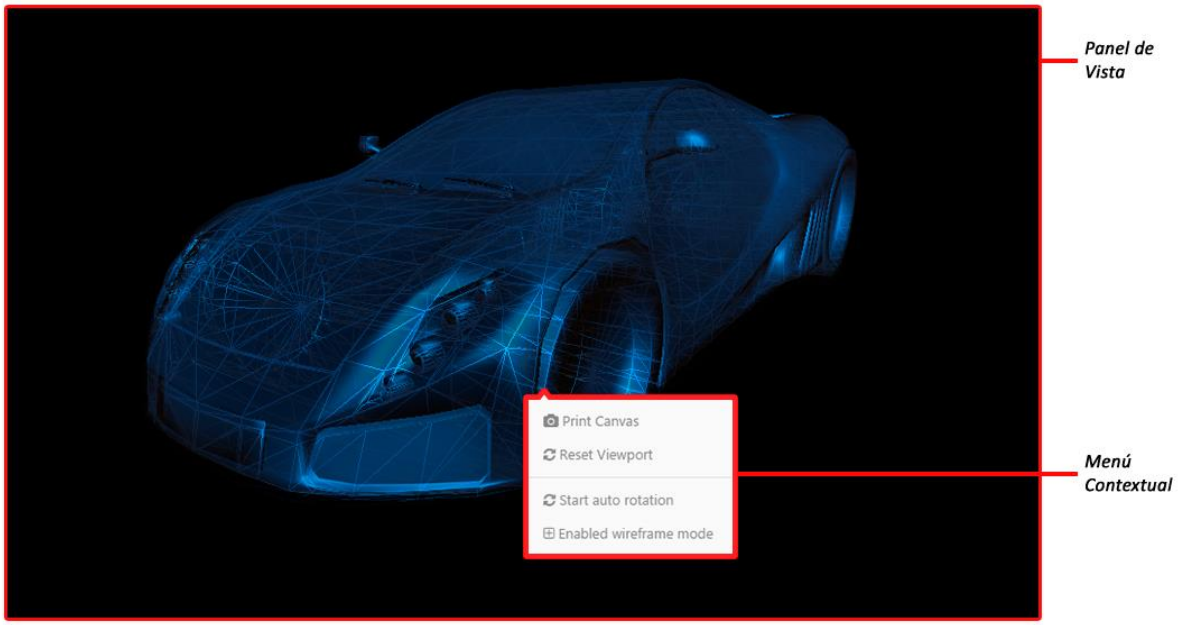


Figura 35 - Indicador de carga de modelo.

Para añadir un nuevo modelo, se cuenta con un área donde se puede seleccionar el modelo arrastrándolo y soltándolo en una región dispuesta para ello, o seleccionándolo desde el selector nativo del navegador. En ambos casos, el archivo se validará y procesara. Para acceder a este componente debe activarse la pestaña *Add Model*.

Una vez seleccionado el modelo se creará de forma automática una escena que desplegará el modelo. Para garantizar el correcto despliegue, el modelo es normalizado y centrado antes de ser mostrado. Mientras el área de pre-visualización esta activa, se activan dos opciones adicionales: Descartar modelo, y detalles del modelo. La escena y tabla de detalles del modelo es mostrado en la **Figura 36**.

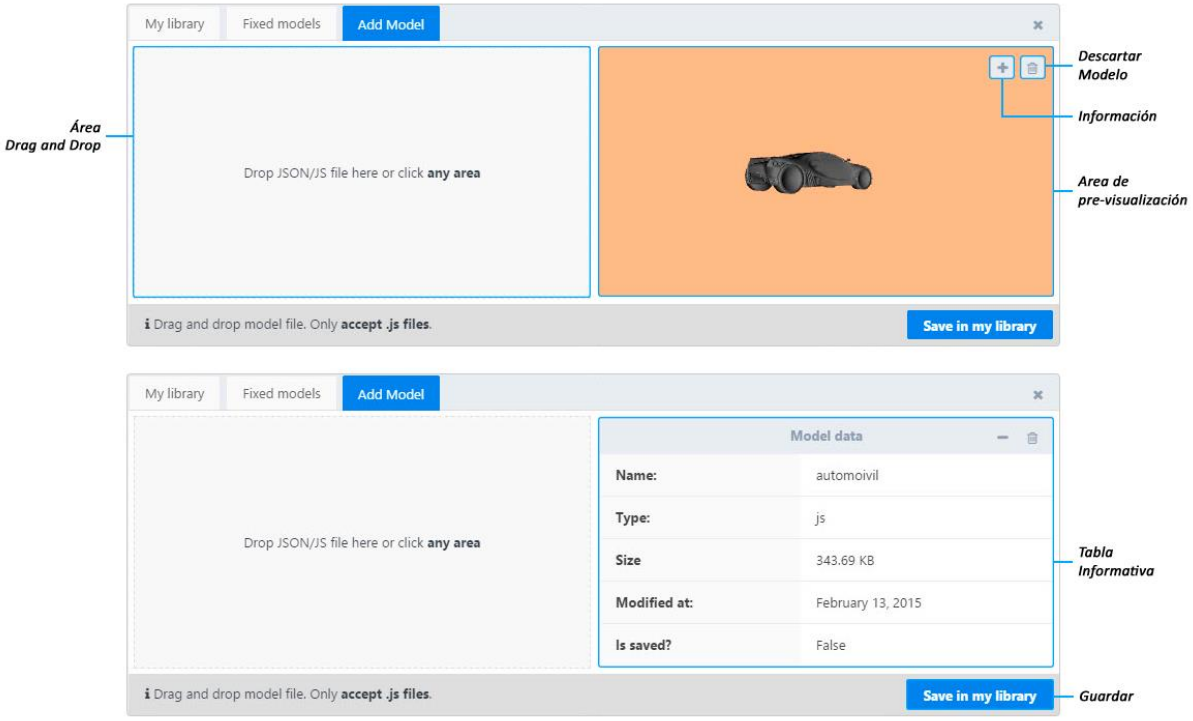


Figura 36 - Carga y descripción de modelos.

Para guardar el modelo, debe hacerse clic en el botón *Save in y library* disponible en la parte inferior derecha del modal. Esta operación inhabilita el modal. Al concluir, muestra de forma automática la biblioteca de modelos del usuario (ver **Figura 37**), destacando el modelo recientemente añadido.

Figura 37 - Lista de modelos añadidos recientemente (Acceso directo).

4.11.7.4 Modal Perfil de usuario

Parte de los datos de usuarios son solicitados al momento del registro. Este modal permite completar los datos faltantes y asociar una imagen al perfil del usuario. Para acceder al modal se debe

hacer desde el menú desplegable, opción *Edit Profile* desde la barra de menús como se muestra en la **Figura 38**.

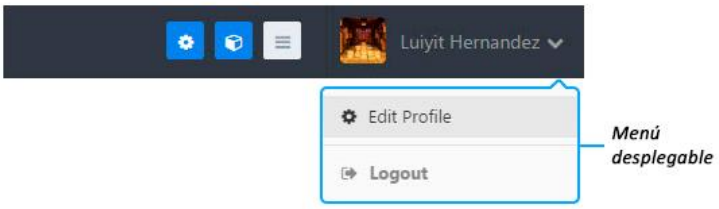


Figura 38 - Acceso al modal de perfil de usuario.

Para cambiar o agregar información faltante, se utiliza el mecanismo *Edit and Place*. En el momento que el usuario hace clic sobre el campo que desea ajustar, se habilita el elemento input correspondiente para que este realice el cambio. Una vez finalizado, al presionar la tecla entrar (*enter*) o hacer clic fuera del elemento, el nuevo valor será tomado enviado al servidor para su actualización. El modal completo se muestra en la **Figura 39**.

Figura 39 - Modal perfil de usuario.

Para asignar una imagen al perfil, se debe hacer clic sobre el *thumbnail*[28] ubicado en la parte izquierda del modal. Al seleccionar una imagen haciendo uso del selector de archivos nativo que se muestra, se despliega un modal superpuesto al actual que permite recortar la imagen antes de asignarla. Al tener seleccionada el área desea se presiona *Crop and Save* y la imagen será actualizada en toda la interfaz de usuario. En el **Figura 40** se ilustra el proceso de actualización de imagen de perfil.

[28] Los *thumbnails* o miniaturas son versiones de imágenes, usadas para ayudar a su organización y reconocimiento

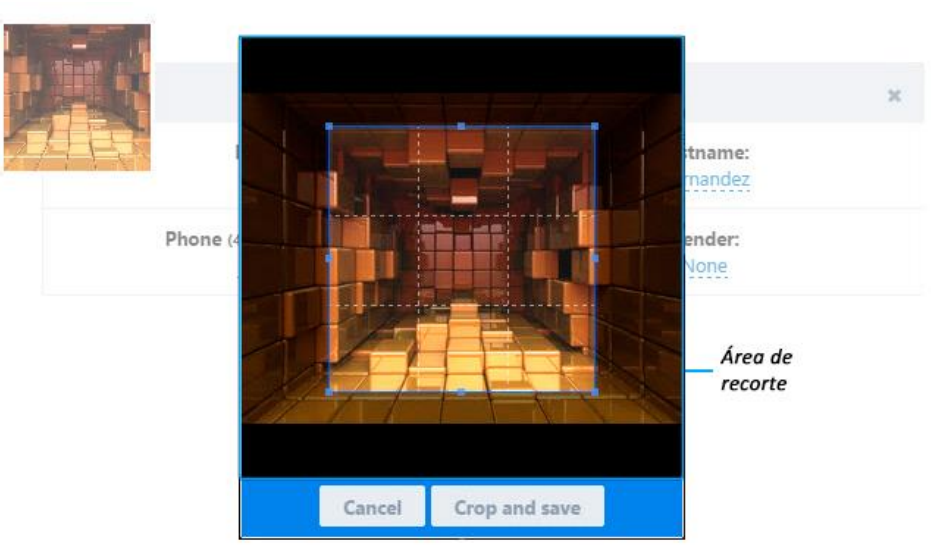


Figura 40 - Modal de recorte de imagen de perfil de usuario.

4.11.8 Página de inicio de sesión y registro

Para hacer uso de la solución se debe iniciar sesión. La **Figura 41** muestra los campos solicitados. De no poseer una cuenta, es posible crear una haciendo clic en el botón *Create New account*.

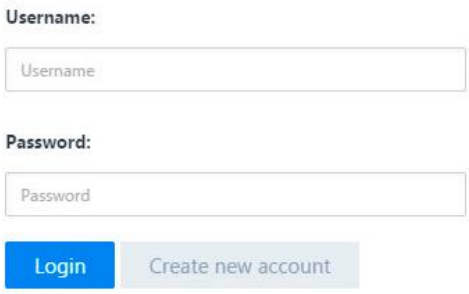


Figura 41 - Página de inicio de sesión.

En el proceso de registro se solicitan principalmente el email y una clave. El email es validado y se garantiza que una dirección de email no sea asociado a más de una cuenta. De resultar los datos correctos se solicita finalmente el nombre de usuario. Este proceso se divide en dos etapas (ver **figura 42**), en una primera interfaz se comprueba la validez de los datos básicos suministrados, y en la segunda parte se cierra el proceso de registro.

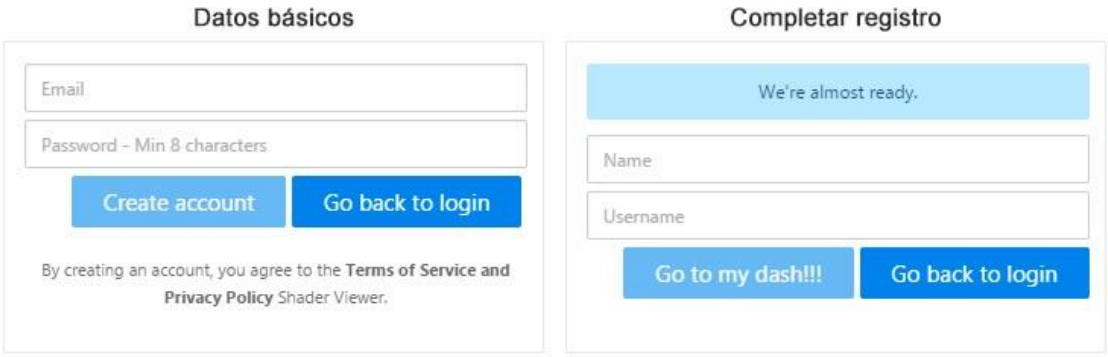


Figura 42 - Interfaz de registro de usuario.

4.11.9 Página de inicio

La página de inicio fue concebida para dar a conocer las características de la solución, en una primera sección se describe su objetivo principal. A continuación se muestran los últimos *shaders* compartidos por la comunidad y el enlace para abrirlos en modo espectador y visualizar el resultado (ver **Figura 43**).

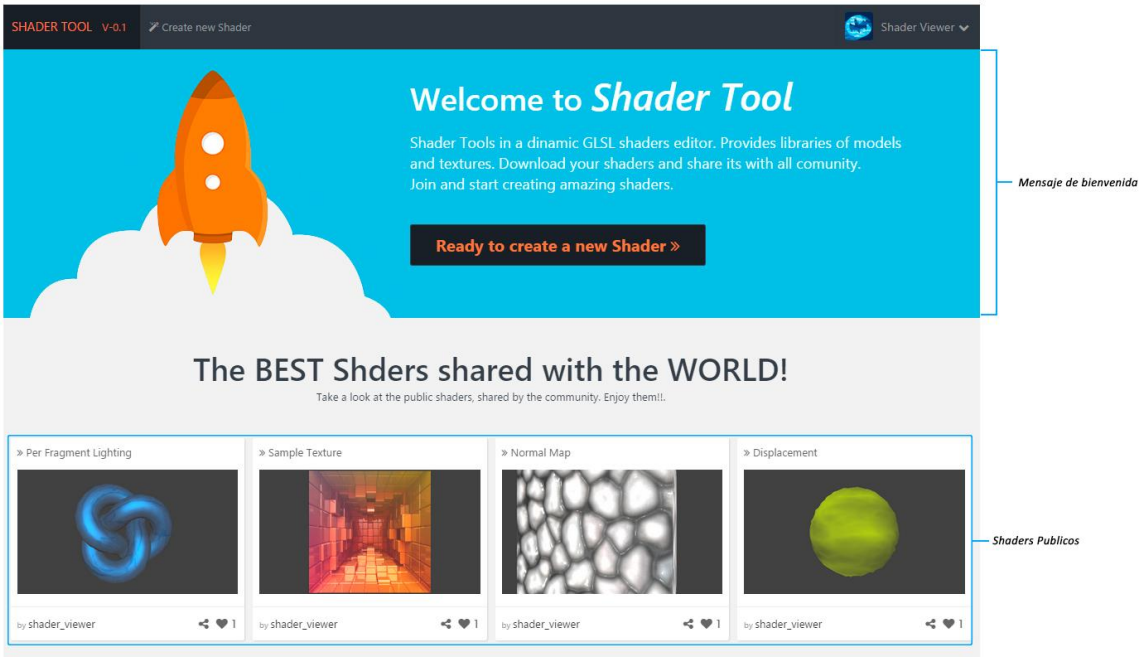


Figura 43 - Página de inicio de la solución.

Adicionalmente, en la **Figura 43** se muestran accesos rápido al editor de *shader* y datos de perfil del usuario. Al final de la página se muestra las características principales de la solución resumidas en 4 ítems. En la **Figura 44** se muestras dichas características: Editor de *shaders*, biblioteca de modelos 3D, biblioteca de texturas y opciones de compartir.

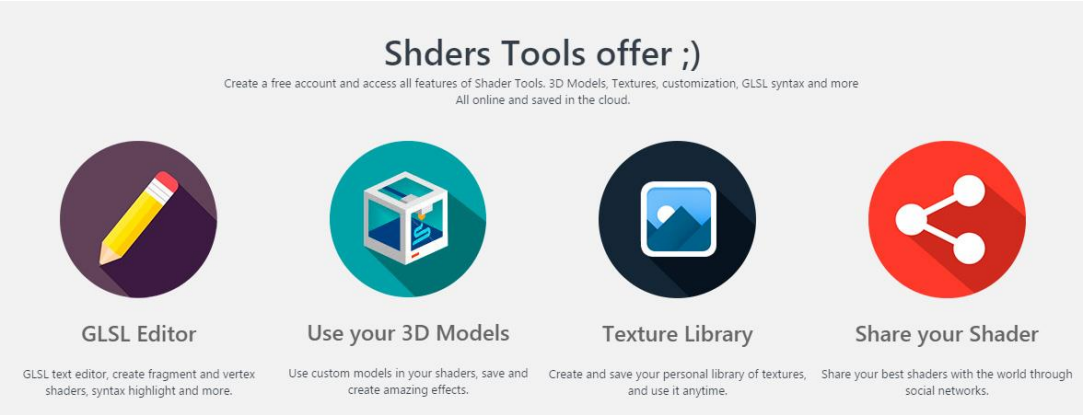


Figura 44 - Características principales de la solución mostradas a final de la página de inicio.

Capítulo 5 Pruebas y Resultados

En este capítulo se presentan las pruebas realizadas para la validación del sistema desarrollado, así como los resultados cuantitativos y cualitativos obtenidos por el uso de la solución. En este trabajo al tratarse de una solución Web, interesa mediar la velocidad de descarga de los diversos recursos, la eficiencia del proceso de compilación y ejecución de los *shaders* y la experiencia de los usuarios en términos de utilidad, facilidad de uso y cumplimiento de los objetivos de la investigación.

5.1 Pruebas y resultados cuantitativos

Estas pruebas tienen como objetivo medir el tiempo y recursos necesarios para la carga de los diversos recursos gestionados en la solución desarrollada en este trabajo. Entre las pruebas se incluyen: velocidad de carga de los modelos en varias configuraciones de red, velocidad de procesamiento de los modelos y tiempo de carga y compilación de los *shaders* usando 4 técnicas de sombreado.

5.1.1 Procesamiento de modelos

Para realizar las pruebas de velocidad de carga y procesamiento se desarrolló un pequeño paquete *Benchmark*[29] que se integra a la solución y reproduce los procesos involucrados en las pruebas. Las clases implementadas registran el tiempo de respuesta del servidor y el tiempo requerido para procesar dicho recuso luego de tenerlo disponible para su uso. El proceso completo involucrado en las pruebas incluye las siguientes etapas:

- Solicitar lista de recursos disponibles (identificadores) del lado servidor para realizar las pruebas. Esta etapa es previa al registro del tiempo.
- Procesar cada identificador de manera individual. Se solicita el recurso completo al servidor.
- Al recibir el modelo, se valida la respuesta y se inicia el procesamiento de los datos.
- Se transforma la cadena de caracteres que contiene información de geometría a una estructura en formato manipulable y estructurado (*JSON.parse*).
- Se procesa la estructura y se crea un objeto propio de la solución manipulable por el *framework* de despliegue (*Three.Geometry*).
- Con la información de geometría se crea un objeto modelo (*Three.Mesh*), el cual posee propiedades adicionales necesarias para el despliegue e interacción con el resto de los elementos de la solución.

[29] Técnica utilizada para medir el rendimiento de un sistema o componente del mismo.

El proceso fue aplicado a una muestra de 10 modelos con distinciones claras en cuanto al número de vértices y triángulos involucrados. Las redes simuladas en las pruebas fueron 2G (450kbps), 3G (1mbps) y 4D (4mbps). En la **Tabla 27** se muestran los resultados obtenidos del proceso completo.

Modelo	Vértices	Triángulos	2G	3G	4G	TTFB[30]
Tetrahedron	4	4	0.45s	0.35s	0.32s	0.34s
Cube	8	12	0.65s	0.36s	0.36s	0.31s
Dodecahedron	20	36	1.05	0.50s	0.44s	0.34s
Cylinder	147	228	3.15s	1.33s	0.89s	0.68s
Plane	256	450	3.80s	1.81s	0.88s	0.36s
Ring	297	512	4.38s	2.54s	0.94s	0.42s
Torus	336	600	7.23s	2.97s	1.46s	0.40s
TorusKnot	768	1536	16.76s	8.02s	3.48s	0.47s
Sphere	1296	2380	27.31s	10.21s	4.85s	0.63s
Text	2292	4572	28.09s	11.88s	4.72s	0.61s
			9.28s	4.02s	1.82s	0.53s

Tabla 27 - Estadísticas de carga y procesamiento de modelos.

Los resultados obtenidos indican un claro crecimiento en el tiempo requerido para descargar y procesar los modelos a medida que la velocidad de la red utilizada es menor. Es claro que la velocidad de descarga recomendada se encuentra entre uno y cuatro megabyte. De igual forma se puede evidenciar que el tiempo de procesamiento de cada modelo no es un factor crítico en el tiempo total, teniendo en cuenta que el promedio obtenido en una red con una velocidad de transferencia alta es de apenas 0.53 segundos.

Es importante destacar que las pruebas se realizaron bajo un entorno red estándar, y se promediaron los resultados obtenidos en 20 simulaciones para cada tipo de red estudiado. Adicionalmente hay que tener en cuenta que en los tiempos indicados no se está restando los milisegundos correspondientes a la fase búsqueda de DNS (DNS Lookup[31]) y tiempo de bloqueo (*Proxy negotioation* o *Establishing Connection*).

Para complementar el análisis de los resultados, se estudió el tiempo de respuesta que ofrece la solución una vez que el recurso ha sido descargado del servidor. En la **Tabla 28** se muestra el

[30] Tiempo de espera del primer byte (*Waitting TTFB*)
[31] Sistema de nomenclatura jerárquica para computadoras, servicios o cualquier recurso conectado a Internet o a una red privada.

tiempo de procesamiento de los datos recibidos para cada modelo analizado, así como la cantidad de memoria RAM[32] que requiere cada objeto. Las fases que conforman el procesamiento del modelo son:

- Validar la información recibida (integridad del objeto).
- Crear la instancia geometry a partir de las cadenas de caracteres recibidas (JSON.Parse).
- Crear objeto Three.Mesh a partir de la geometría resultante.

Al crear la instancia del objeto, queda disponible para su uso en cualquier otro proceso. Teniendo en cuenta que son varias las rutas que pudiera tener el modelo (despliegue principal, pre-visualización, asociación, listar datos básicos para su selección, entre otros), la prueba de procesamiento se evalúa hasta este punto, considerando única y exclusivamente el tiempo requerido para transformar los datos recibidos desde el servidor, en un objeto Javascript válido.

Modelo	Vértices	Triángulos	Memoria	Procesamiento
Tetrahedron	4	4	27 KB	0.42s
Cube	8	12	78 KB	0.45s
Dodecahedron	20	36	231 KB	0.49s
Cylinder	147	228	1.4 MB	0.45s
Plane	256	450	2.8 MB	0.46s
Ring	297	512	3.3 MB	0.48s
Torus	336	600	3.8 MB	0.74s
TorusKnot	768	1536	9.8 MB	0.55s
Sphere	1296	2380	15.3 MB	0.61s
Text	2292	4572	29.1 MB	0.65s

Tabla 28 - Estadística de procesamiento de modelos.

Las pruebas realizadas evidencian que la velocidad de la red utilizada para acceder a la solución tendrá un alto impacto en la velocidad de respuesta. EL mayor porcentaje de tiempo en obtener un recurso corresponde al proceso de descarga, teniendo en cuenta que en promedio el valor TTFB es de 0.5 segundos. Durante este tiempo de espera por parte del cliente, se llevan a cabo entre otros procesos, la recepción de la petición, su procesamiento y respuesta. Manteniendo un tráfico bajo, la solución del lado servidor es capaz de atender las peticiones en fracciones de segundo.

[32] (*Random-Access Memory, RAM*) se utiliza como memoria de trabajo de computadoras para el sistema operativo, los programas y la mayor parte del software.

Por otra parte, el procesamiento de los modelos luego de recibir el 100% de los datos oscila entre 0.42 segundos y 0.74 segundos, lo cual indica que en conjunto (cliente y servidor) la solución requiere en promedio de 1 segundo para procesar cualquier modelo.

5.1.2 Procesamiento de *Shaders*

El proceso encargado de gestionar y procesar el código de los *shaders* es uno de los principales dentro de la solución. Las pruebas asociadas a los programas GLSL se dividen en dos partes. La primera parte registra el tiempo de procesamiento del *shader*, desde el momento en que se recibe toda la información desde el servidor, hasta crear las estructuras necesarias y compilar el código. Los resultados varían de acuerdo a las capacidades de procesamiento tanto del procesador (CPU – Unidad de Procesamiento Central) como de la tarjeta gráfica (GPU – Unidad de Procesamiento Gráfico) del equipo que ejecuta la solución del lado del cliente.

Para validar el comportamiento de la solución en varios escenarios, las pruebas se llevaron a cabo en tres configuraciones de *hardware* distintas: GeForce GTX 750, AMD Radeon HD 8550G y ARM Mali-400. En la **Tabla 29** se muestran los valores obtenidos.

Shader	# líneas	Uso de texturas	GeForce GTX 750	AMD Radeon HD 8550G	ARM Mali-400
Basic	9	✗	0.0043s	0.0055s	0.0177s
Sample Texture	15	✓	0.0047s	0.0183s	0.0195s
Displacement	35	✓	0.0085s	0.0094s	0.0257s
Per Fragment Lighting	45	✓	0.0077s	0.0122s	0.0317s
Normal Mapping	153	✓	0.0153s	0.0278s	0.0613s

Tabla 29 - Estadísticas de Procesamientos de *shaders*.

Los *shaders* analizados fueron seleccionados con el objetivo de atender los aspectos principales de los *shaders* desarrollados con mayor frecuencia. La función de cada efecto se detalla a continuación:

- ***Basic***: *Shader* con operaciones básicas. Se procesa de forma simple la posición de cada vértice y se asigna un color estático a cada fragmento. En el Efecto más básico.

- ***Sample Texture***: Se incluye el manejo de un mapa de bits en el programa de fragmento. Para definir el color de cada fragmento se hace uso de las coordenadas de texturas y se extrae la muestra directamente de la textura usada.

- ***Displacement***: El objetivo principal del *shader* es contar con animación de vértice y color. Se configuran variables uniformes y atributos a cada vértice para animar la geometría. Esto

implica enviar en cada cuadro los datos actualizados a la tarjeta gráfica, lo cual supone un esfuerzo mayor.

- ***Per Fragment Lighting***: Técnica de iluminación en la cual la iluminación se basa en cada fragmento de la imagen final. Se incluyen componentes de luz difuso y especular, así como también el uso de los vectores normales para cálculos de incidencia de luz sobre cada fragmento.

- ***Normal Mapping***: Permite iluminar y simular relieve en superficies planas. Para ello se hace uso de texturas especiales (Normal Map y Specular Map) y los vectores *Tangent* y *Binomial*. Aumenta el realismo de los objetos desplegados y requiere de un procesamiento mayor ya que se requieren realizar cálculos adicionales al sombreado tradicional para lograr la ilusión de relieve.

Los resultados obtenidos al procesar cada *shader* (**Tabla 29**) muestran que tanto en computadores personales como en dispositivos móviles el tiempo requerido por la solución está por debajo de una décima de segundo. Teniendo en cuenta la velocidad de carga y procesamiento de los modelos, esta cifra se encuentra en un rango aceptable, ya que en el peor de los casos el *shader* será procesado en el mismo tiempo que los modelos. Es importante destacar que las pruebas realizadas muestran una relación entre el número de líneas y tiempo requerido para ser procesadas. Esto se debe al número de operaciones aritméticas involucradas en el *shader*, lo cual nos indica que las optimizaciones realizadas al código de cada programa es un factor importante a considerar.

Otro valor importante asociado a los *shaders* es la velocidad con la que el despliegue se realiza. Las imágenes por segundos (*Frame Per Seconds*) es la medida de la frecuencia a la cual un reproductor de imágenes reproduce distintos fotogramas (*frames*). Estos fotogramas están constituidos por un número determinado de píxeles que se distribuyen a lo largo de una red de texturas. La frecuencia de los fotogramas es proporcional al número de píxeles que deben generarse, incidiendo en el rendimiento del ordenador que los reproduce. En la **Tabla 30** se muestra la velocidad de despliegue para cada *shaders* analizado.

Shader	GeForce GTX 750	AMD Radeon HD 8550G	ARM Mali-400
Basic	60 FPS	60 FPS	60 FPS
Sample Texture	60 FPS	60 FPS	56 FPS
Displacement	60 FPS	57 FPS	35 FPS
Per Fragment Lighting	60 FPS	60 FPS	28 FPS
Normal Mapping	60 FPS	60 FPS	25 FPS

Tabla 30 - Pruebas de rendimientos de la solución en diferentes configuración de *Hardware*.

Los resultados arrojados por las pruebas de rendimiento evidencian que tanto el procesamiento de los *shaders* como la velocidad de despliegue se encuentran en el rango óptimo (24FPS) para una solución de procesamiento de gráficos generados por computador. Todos los *shaders* fueron aplicados a un modelo de alta definición, el cual está formado por 2200 vértices y 4500 triángulos.

5.1.3 Editores en línea vs *Shader Tool*

En la solución desarrollada se incorporaron diversas herramientas y módulos con el objetivo de consolidar y unificar criterios en el ámbito de los editores de *shaders online*. A continuación se evalúa cada aspecto de la solución en base a las características consideradas inicialmente, y con las cuales se evaluaron a los editores de *shaders* analizados.

5.1.3.1 Editor de texto

En la **Tabla 31** se muestra la comparativa en relación al panel editor entre el promedio obtenido de los editores de *shader* analizados con anterioridad y la solución desarrollada.

	Shader Tool	Promedio
Auto completado	✗	✗
Validador de error	✓	✓
# de línea	✓	✓
Snippets	✗	✗
Descargar código	✓	✗
Indicador gráficos	✓	✓
Editor de parámetros in	✓	✗
Comodidad de edición	5/6	3/6
Auto guardado y despliegue	✓	✓
Colapso se secciones	✓	✓
Búsquedas	✓	✗

Tabla 31 - Comparativa de aplicaciones analizadas con la solución *Shader Tool* (Editor de texto).

5.1.3.2 Canvas y utilidades

En esta sección se evalúan las características de despliegue de la solución al momento de la implementación de los *shaders*. En la sección se considera las características de la biblioteca de recursos, opciones de configuración de despliegue, información disponible referente al *shader* desarrollado, entre otras (ver **Tabla 32).**

	Shader Tool	Promedio
Indicador FPS	✔	✖
Ejemplos / Modelos	6/6	3/6
Transformaciones básicas	✔	✖
Detalles de despliegue	✔	✖
Manejo de eventos	✖	✖

Tabla 32 - Comparativa de aplicaciones analizadas con la solución *Shader Tool* (Área de despliegue y otras).

5.1.3.3 Despliegue (*Render*)

Es de vital importancia evaluar la solución en cuanto a las capacidades de despliegue y herramientas disponibles en el proceso de programación de *shaders.* En la **Tabla 33** se muestra la comparativa entre las herramientas incluidas en *Shader Tool* y los editores analizados.

	Shader Tool	Promedio
Contexto 2D/3D	3D	3D
Uso de texturas	✔	✖
Ajuste Modo de despliegue	✔	✖
Ajuste Cara a desplegar	✔	✖
Despliegue de normales/BB	✖	✖
Ajuste cambiar Sombreado	✖	✖
***Shaders* (G-F-V)**	F-V	F-V
Conf. de Cámara	✔	✖
Manejo de Luces	✖	✖
Formatos admitidos	JSON	Privado
Biblioteca de recursos	✔	✖
Galería de efectos	✔	✖

Tabla 33 - Comparativa de aplicaciones analizadas y *Shader Tool* (Proceso de Render).

5.1.3.4 Experiencia de usuario y usabilidad

Finalmente se evalúan aspectos relacionados con la interfaz de usuario, disposición de los elementos, componentes informativos, adaptabilidad, opciones sociales, entre otros aspectos involucrados en experiencia de usuario. En la **Tabla 34** se presentan dichos aspectos.

	Shader Tool	Promedio
Tipo de navegación	Compuesta	Ninguna
***Layout* flexible**	✔	✔
Interacción con periféricos de entrada/Salida	✔	✖
Retroalimentación Visual	✔	✖
Zoom óptico	✖	✖
Compartir	✔	✖
Selector de archivo	✔	✖
Controles	✔	✔
Menú contextual	✔	✖
Pantalla completa	✔	✔
Manejo de sesiones	✔	✖

Tabla 34 - Comparativa de aplicaciones analizadas y *Shader Tool* (Experiencia de usuario y usabilidad).

5.1.3.4 Análisis de los resultados del estudio comparativo

Los resultados obtenidos en el estudio comparativo entre la solución desarrollada y los *shaders* analizados muestran mejoras considerables en todos los tópicos estudiados por parte de la solución *Shader Tool*. En el aspecto del editor de texto, se tomaron en cuenta características importantes como la descarga de los *shaders* y el uso de parámetros de entrada lo que influyo una mejor valoración.

En cuanto al tópico del elemento HTML canvas y el despliegue, se obtuvo un porcentaje de mejora de aproximadamente 65%, teniendo en cuenta que de las diecisiete características analizadas, once presentaron mejoras en la solución desarrollada.

La experiencia de usuario fue el tópico con mayor porcentaje de mejoría. De acuerdo a los resultados obtenidos nueve de diez aspectos estudiados (90%) fueron tomados en cuenta en la solución, entre los que se encuentra la posibilidad de compartir el contenido generado y de restaurar el ambiente de desarrollo en cualquier momento, lo que constituye mejoras significativas en el flujo de desarrollo de los usuarios.

5.1.4 Velocidad de carga

La solución desarrollada esta por un conjunto de archivos en formato Javascript, CSS, HTML, entre otros. Por la naturaleza de la arquitectura utilizada, todo el contenido y archivos necesarios para el funcionamiento de la solución deben ser descargados de forma íntegra. De igual manera, los diversos módulos deben ser iniciados y enlazados.

Para estimar el rendimiento de la solución en cuanto a tiempo de inicio, se realizaron simulaciones de descarga combinando diversas plataformas de sistemas operativos actuales y navegadores comerciales. En la **Tabla 35** se muestra el tiempo requerido en el cual la aplicación se inicia y se encuentra disponible para su uso.

Para la ejecución de todas las pruebas se utilizó una conexión de rendimiento promedio (Móvil 3G - 780kbps/330kbps) y las peticiones se realizaron desde un equipo ubicado en Dallas, VA USA. El objetivo de realizar las pruebas desde un equipo que este significativamente distante del servidor que aloja la solución, es generar los resultados bajo condiciones que representen una exigencia mayor en cuanto al nivel de optimización de la solución.

Dispositivo - Sistema Operativo	Browser	Despliegue inicial	2do despliegue	1er Byte
Desktop - Microsoft Windows 8	Chrome 40	32.451s	2.505s	1.079s
Desktop - Microsoft Windows 8	Firefox	35.880s	1.707s	0.681s
Desktop - Microsoft Windows 8	IE11	28.965s	0.471s	1.077s
Motorola Moto G - Android OS 4.3 (Jelly Bean)	Chrome	34.385s	4.703a	1.452s
Nexus 5 - Android OS 4.1 (Jelly Bean)	Chrome	33.962s	4.197s	1.477s
Nexus 7 Landscape - Android OS 5.0 (Lolipop)	Chrome	33.865s	3.849s	1.201s
iPhone 4.0 - OS 5.1	Safari	23.392s	4.835s	0.00s
		27.207s	2.823s	0.736s

Tabla 35 - Estadísticas de la velocidad de inicio de la solución.

De acuerdo a los resultados obtenidos, La solución tiene un promedio de inicio de 27.20 segundos empleando una velocidad de conexión promedio. Al evaluar los resultados se puede notar una mejora considerable en dispositivos que permiten paralelizar un mayor número de peticiones. En el caso particular del dispositivo iPhone 4.0, el número de conexiones atendidas en un momento dado es de veintidós. Lo cual favorece significativamente en la inicialización de la solución.

Por otra parte, al evaluar los resultados de inicio se visualiza una notoria mejora en el despliegue de la aplicación en accesos posteriores a la carga inicial. En todos los casos de estudio se obtiene una mejora superior al ochenta por ciento. También se muestra que el tiempo necesario se mantiene estable en los diversos dispositivos y sistemas operativos.

Para ilustrar la composición final de la solución y la división de los recursos descargados en las pruebas se muestra en la **Tabla 36** el resumen porcentual de cada tipo de archivo descargado y su tamaño medido en kilobytes.

Tipo de Archivo	Peticiones	Porcentaje Peticiones	Tamaño (Kilobyte)	Porcentaje Tamaño
JS	31	68,90%	2.292	82,70%
Imágenes	6	13,30%	122	4,40%
CSS	2	4,40%	221	8%
Fuentes	2	4,40%	128	4,60%
HTML	2	4,40%	4	0%
Otros	2	4,40%	3	0%

Tabla 36 - Desglose de la descarga inicial por tipo de archivo.

El desglose también puede visualizarse gráficamente en el **Figura 45**. Ambos indicadores muestran que la solución está conformada principalmente por archivos Javascript. Aunque la solución contempla el uso de aproximadamente 15 documentos HTML, estos son gestionados usando un servicio de optimización llamada *TemplateCache*. El funcionamiento del servicio se basa en almacenar en memoria una función Javascript que da acceso al contenido de cada vista. Cada archivo HTML ahora puede ser accedido mediante una palabra clave.

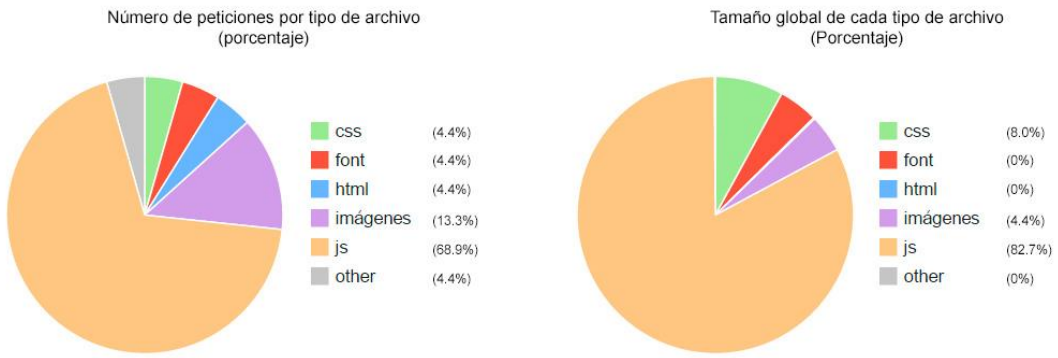


Figura 45 - Gráfico de la representación de cada tipo de archivo referente al número de peticiones y tamaño.

Basado en ese concepto, cada archivo HTML es representado por un archivo JS el cual es incluido como parte de la aplicación. Este procedimiento incremente significativamente el tiempo de respuesta de la aplicación una vez que el usuario accede a los diferentes módulos, ya que todas las vistas fueron descargadas en la inicialización de la aplicación.

5.2 Pruebas y resultados cualitativos

Las pruebas cuantitativas se basaron en recopilar las distintas experiencias de los usuarios al utilizar la solución en términos de facilidad de uso, utilidad y cumplimiento de los objetivos de la investigación.

El instrumento diagnóstico utilizado fue una encuesta en torno a ocho (8) preguntas contenidas en cuatro tópicos principales:

1) Utilidad de la interfaz gráfica de usuario.
2) Facilidad en la creación de *Shaders*.
3) Evaluación cualitativa de los recursos disponibles.
4) Utilidad de la solución en términos de tiempo, esfuerzo y recursos utilizados.

La síntesis de los resultados obtenidos en la encuentra agrupados por tópicos se puede visualizar en la **Figura 46**. El rango de valoración fue establecido entre 1 y 5, donde 5 representa un desempeño óptimo dentro de las características del tópico y 1 representa deficiencias en uno o varios aspectos asociadas.

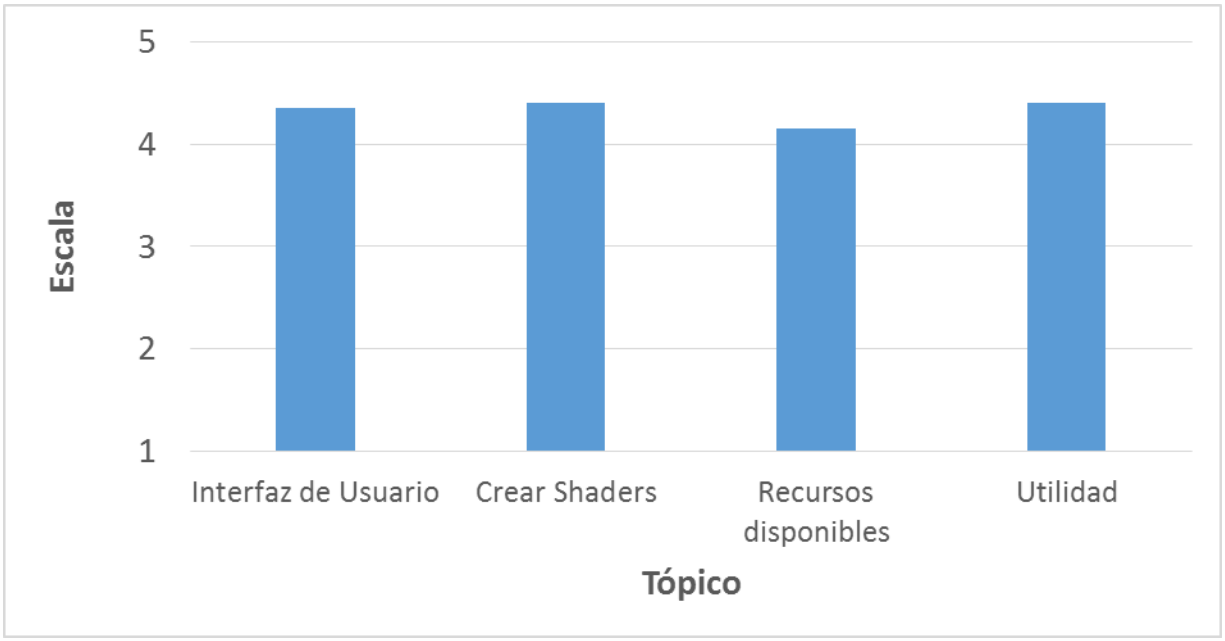


Figura 46 - Valoración obtenida en cada tópico evaluado.

La encuesta fue realizada a estudiantes de la escuela de Computación de la Universidad central de Venezuela. Los encuestados cursan la asignatura Inducción a la Computación Grafica y ya cuentan con los conocimientos técnicos involucrados en el desarrollo de *shaders* en el Lenguaje GLSL.

5.2.1 Análisis de los resultados

A continuación, se detallan y analizan los resultados obtenidos:

- **Utilidad de la interfaz gráfica de usuario**: en este rubro se evaluó la interacción humano-computador, la usabilidad y la curva de aprendizaje de la solución y de todas sus funcionalidades.
    - Los resultados de la encuesta indican que todas las herramientas interactivas proporcionadas por la interfaz gráfica de usuario facilitan en gran medida el uso de la solución, ya que las mismas permiten ajustar el área del editor y *viewport* a voluntad, acceso a las diferentes opciones a través de múltiples interfaces y la posibilidad de ejecutar comandos directamente desde atajos de teclado.
    - Los resultados revelan que el usuario considera que la información brindada por el sistema en de gran ayuda al momento de desarrollar el *shader*. Gran parte de los encuestados coinciden que los diferentes módulos presentes en la solución como el manejo de texturas, modelos, detalles de *shaders*, entre otros, permiten disponer de información y recursos de forma rápida y cómoda, ya que todos los elementos se encuentras disponibles y enlazados en una misma área de trabajo.

- **Facilidad en la creación de *Shaders***: en este tópico se evaluó el procedimiento entorno a la creación de *Shaders* y la construcción de biblioteca de recursos del usuario.
    - A través de la encuesta se pudo validar que el proceso de desarrollo de *shaders* es considerado práctico y rápido por los usuarios. En todo momento se conoce claramente sobre cual programa GLSL se está trabajando, la solución permite rápidamente cambiar entre los programas y los resultados del *Shaders* son mostrados rápidamente en el panel de despliegue.
    - Adicionalmente los usuarios encuestados indicaron que el nivel de personalización de la interfaz al momento de crear los *Shaders* es considerablemente alto, ya que la solución ofrece características varias para enfocar la atención en el panel requerido Opciones como pantalla completa, redimensión automática del área de despliegue y visualización del efecto de forma individual conforman las herramientas de mayor impacto en este módulo.

- **Evaluación cualitativa de los recursos disponibles**: En esta sección de la encuesta se evalúo la calidad y diversidad de recursos disponibles en la solución de forma inmediata para los usuarios.
    - Las respuestas obtenidas revelan que el hecho de que la solución permita a los usuarios usar una biblioteca de modelos, texturas y *shaders* estándar da un valor agregado a la herramienta. Esto les permitió a los usuarios crear *shader* y concentrarse exclusivamente en la programación y no en los recursos necesarios.

- **Utilidad de la solución en términos de tiempo, esfuerzo y recursos utilizados**: en este apartado de la encuesta se evaluó la impresión de los usuarios en relación a la velocidad de respuesta de la solución.
    - Los resultados obtenidos revelan que la velocidad de carga de la solución es el aspecto más cuestionado. Los usuarios no coinciden en una calificación uniforme en este rubro. Una parte de los encuestados indican que el desempeño es adecuado y otro grupo que es bajo o ineficiente. Sin embargo, hay que considerar que la calificación obtenida está asociada a la velocidad de conexión que posee el usuario.
    - Por otra parte, se pudo validar que los indicadores informativos que indican el estado de los procesos son de gran ayuda para los usuarios ya que en momentos donde la respuesta puede demorar segundos, el usuario tiene conocimiento de que el proceso va en marcha.

5.2.2 Resultados en ejecución

Esta prueba describe el proceso completo para la creación de un *shaders* empleando los diferentes módulos y herramientas presentes en la solución. De esta forma se validan los resultados obtenidos en la encuesta realizada, y se presenta a través del flujo de la solución las facilidades de uso y lo intuitiva que resulta la interfaz gráfica.

En primera instancia se debe acceder a la solución del lado cliente desde cualquier navegador que brinde soporte para los diferentes frameworks de desarrollo utilizados (consultar **Tabla 8**). Una vez la página de inicio cargue, se debe acceder al área de registro (ver **Figura 47**).

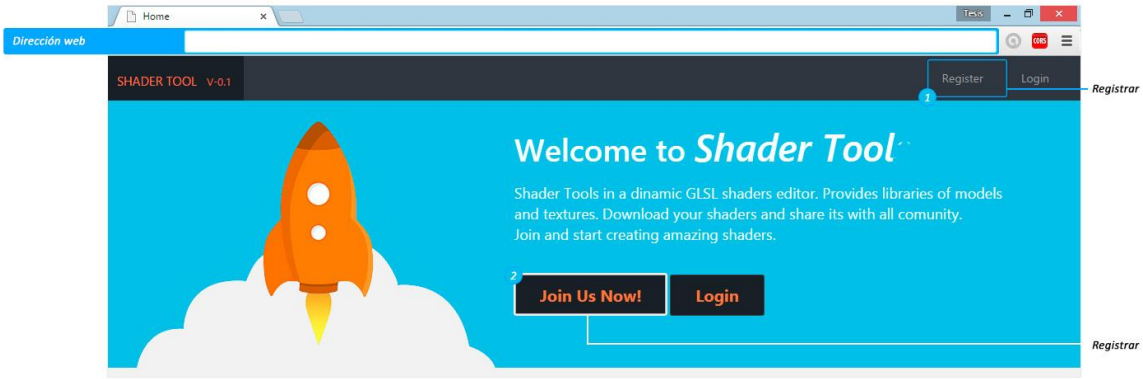


Figura 47 - Acceso disponible para el registro de usuarios.

En la página de registro debe proveer los datos solicitados en el formulario, al completarse se hace clic en el botón “Create account” (ver **Figura 48**). Los datos serán validados por el servidor Web. De resultar ser un email válido el sistema mostrará una notificación exitosa y da acceso al formulario final de registro.

Figura 48 - Interfaz para crear una cuenta nueva.

Para finalizar el registro se debe completar los campos nombre y nombre de usuario, el cual también debe ser único (ver **Figura 49**). Hay que tener en cuenta que en ambos formularios el botón de procesar estará deshabilitado mientras los datos sean proporcionados o sean válidos.

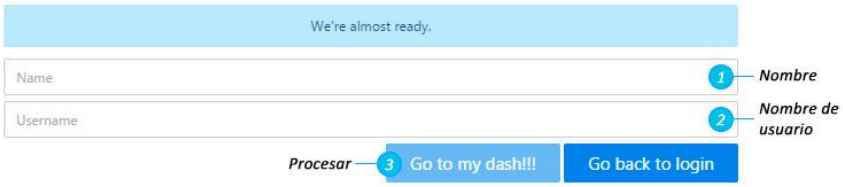


Figura 49 - Interfaz de solicitud de datos finales para crear una cuenta nueva.

El proceso de registro hará el inicio de sesión de manera automática y configura la solución de manera automática para dar acceso a todos los módulos. De la misma forma, los recursos mostrados serán única y exclusivamente los creados por el usuario actual. En este punto ya es posible hacer el desarrollo de un *shader*, para acceder al editor basta con hacer clic en la opción "Editor GLSL" que se encuentra disponible en la barra de menús o desde el botón en el área del banner principal mostrado en la **Figura 50**.

Figura 50 - Acceso al editor GLSL.

En la página del editor se mostrara por defecto un *shader* básico con sus dos programas asociados. Cada uno de ellos sirve como plantilla inicial para el desarrollo. Para crear un nuevo *shader* y almacenarlo en la biblioteca del usuario se cuenta con dos opciones. La primera en desde el menú desplegable en la barra de menús haciendo clic en la opción "*New Shaders*", o usando el comando de

teclado “ctrl + shift + s”. Al mostrarse el modal (ver **Figura 51**) correspondiente debe proveerse un nombre, la plantilla a utilizar y el modelo asociado. Se oprime el botón “*Create Shader*” y la solución registra los datos suministrados en el servidor y los asociara a los datos de usuario de la sesión actual.

Figura 51 - Flujo e interfaces involucradas en la creación de un *shader*.

Al crearse el *shader* los diferentes módulos se configura en torno a esté. Automáticamente los programas *vertex shader* y *fragment shader* se muestran en el área del editor, y el modelo es descargado y desplegado en el panel de vista (*viewport*). Solo un programa está activo en un momento determinado, para hacer el cambio se debe hacer clic en el control *togle* destinado para hacer el cambio (ver **Figura 52**).

Figura 52 - Control de cambio de programa de *shader*.

Para compilar y ejecutar el código del *shader* (luego de la edición) se presiona el botón “*play*”. De no contener errores, el código es ejecutado y desplegado. De manera automática (en background) se guardan de forma permanente los cambios realizados en el *shader*, y se hacen capturas periódicas del área de despliegue para guardarla como *thumbnail* del efecto.

Entre las herramientas adicionales disponibles en la solución se encuentra la biblioteca de modelo y de texturas. Para cambiar el modelo usado, se debe hacer clic en el botón con el icono de cubo. Este da acceso al modal de la biblioteca de modelos (ver **Figura 53**). Para utilizar los modelos disponibles se hace clic en la pestaña “*Fixed models*”, y se oprime el botón al final de la fila del modelo seleccionado.

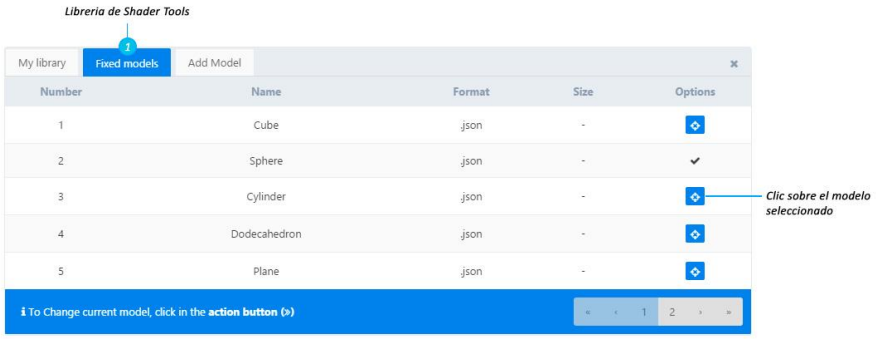


Figura 53 - Modal de selección de modelo.

Para incluir el uso de textura debe incluirse las imágenes necesarias en el modal "Texture Selector" mostrado en la **Figura 54**. Se accede a este modal a través del botón textura (icono *picture*), en él se listarán todas las imágenes disponibles. Haciendo clic en el *checkbox* de cada fila, la solución registrará una variable uniforma del tipo "*sampler2D*" en el material asociado al *shader* con la información asociada a la imagen (mapa de bits).

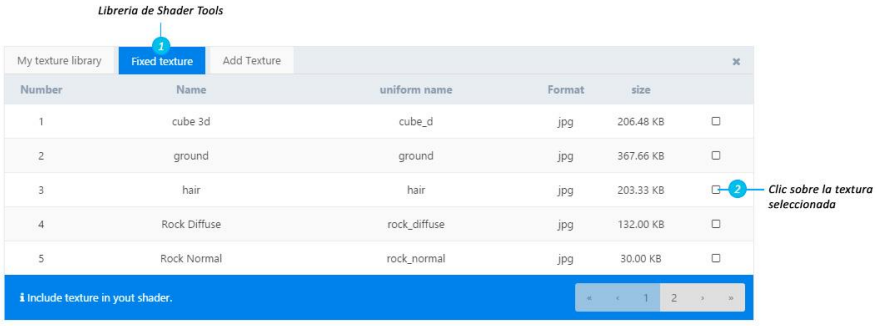


Figura 54 - Modal de selección de texturas.

Entre los datos mostrados en el modal de selección de textura (**Figura 54**), se muestra el nombre con el cual la textura es registrada. Este nombre debe ser usado para declarar la variable en el *shader* de fragmente y así poder acceder al mapa de bits.

Para complementar y agilizar el desarrollo del *shader*, se encuentra disponible opciones de configuración de la cámara, parámetros de despliegue, pantalla completa y detalles del *shader* actual. Todas estas opciones son accedidas desde la barra de menús.

Capítulo 6 Conclusiones y Trabajos Futuros

En este trabajo se ha desarrollado un sistema Web integrado que permite desarrollar programas de sombreado para OpenGL, haciendo uso de *frameworks* avanzados de despliegue y herramientas de marcado y estilo. A continuación se listan las conclusiones y los trabajos futuros más importantes producto de esta investigación.

5.1 Conclusiones

Durante la investigación se analizaron todas las variables y condiciones que intervienen durante el desarrollo de programas GLSL en un ambiente de Web. Con la información obtenida se estudiaron las limitaciones de los editores actuales disponibles en la actualidad y se encontró que se debía desarrollar un sistema integrado que integrará tecnología Web de vanguardia y los diversos componentes involucrados en el proceso a través de una interfaz gráfica de usuario.

El análisis de la interacción del usuario con la solución desarrollada y los diversos principios de diseños como la jerarquía visual, legibilidad e integridad, fueron aplicados para reducir la curva de aprendizaje de la solución y de todas sus funcionalidades. De acuerdo a los resultados obtenidos en las encuestas realizadas, se encontró que el conjunto de componentes e interfaces gráficas que se diseñaron e implementaron permiten al usuario interactuar con el sistema de manera sencilla, facilita el desarrollo de los *shaders* y su asociación con elementos como texturas, modelos y variables de control.

Los servicios *Models* y *Textures* de la solución Web permiten cargar y procesar los recursos principales usados en la fase de despliegue del *shader*. Los indicadores de carga, controles de páginas y carga fragmentada de recurso en los modales que despliegan la lista de modelos 3D y texturas, fueron introducidos como mejoras para ayudar a la velocidad de carga e interacción con los usuarios.

La solución es capaz de cargar y desplegar modelos 3D haciendo uso de archivos en formato JSON. Gracias a esto los usuarios cuentan con una biblioteca pública que puede ser utilizada en sus *shaders*. De la misma forma, cada programador puede crear y gestionar sus modelos de manera independiente. Los resultados de la encuesta arrojaron que los usuarios se encontraron muy satisfechos con esta característica. Igualmente, esto hace posible que en el futuro se agreguen nuevos modelos.

El sistema posee un módulo para la validación del código escrito por los usuarios, el cual compila cada programa (V*ertex Shader* y *Fragment Shader*) de manera individual y arroja los posibles errores que estos puedan tener. Esto permite que el usuario se informe de manera rápida de los fallos que presenta sus *shaders* antes de enviarlos a la unidad de procesamiento gráfico.

Con la finalidad de proveer herramientas útiles e innovadoras dentro del ámbito Web, se incorporaron biblioteca y módulos para el manejo de comandos de teclado, visualización de la solución en pantalla completa, descarga de los *shaders* desarrollados, uso de menús contextuales y adaptación de los elementos gráficos de acuerdo a la resolución del dispositivo de despliegue.

Los resultados de la encuesta también indican que la información brindada en todo momento, la simplicidad existente en la interacción del usuario con todos los componentes e interfaces gráficas, la posibilidad de crear biblioteca de recursos personalizadas, y el hecho de poder compartir el contenido generado, son factores por los cuales se presenta la solución como una posible herramienta de entrenamiento y formación interactiva para la comunidad de estudiantes y programadores del área de computación gráfica.

También se realizaron diferentes tipos de pruebas cuantitativas con el objetivo de determinar la eficiencia en el uso de los recursos disponibles por parte de la solución desarrollada en este trabajo. Las pruebas incluyeron:

- Velocidad de carga y procesamiento de modelos 3D: Los resultados indicaron que en promedio el tiempo requerido para procesar un modelo oscila entre 0.4 y 0.7 segundos, lo que implica que la porción de tiempo del proceso completo (descarga y procesamiento) para cualquier modelo es asignado a la fase de conexión y descarga del recurso.

- Velocidad de procesamiento de *shaders*: Los resultados reflejan que para procesar los *shaders* se requiere en promedio 0.1 segundos. El proceso incluye la fase de validación y la creación de la instancia del efecto.

- Velocidad de cuadros desplegados por segundos (FPS): Se realizaron pruebas de desempeño medida en cuadros por segundos de los diferentes *shaders* implementados en la solución. Los resultados varían de acuerdo a las prestaciones de la unidad de procesamiento gráfico. Para el modelo de unidad de procesamiento gráfico GeForce GTX 750 y AMD Radeon HD 8550G el promedio de despliegue estuvo por encima de 60 FPS. Por otra parte, para el modelo de tarjeta ARM Mali-400 (disponible en dispositivos móviles) las pruebas indicaron una velocidad promedio de 30 FPS.

5.2 Trabajos Futuros

El sistema puede ser adaptado a futuro para permitir la incorporación de un mayor número de parámetros asociados al proceso de despliegue; esto es, incluir la posibilidad de manipular parámetros de transparencia o blending en los materiales asociados a los modelos, uso de una fuente de luz configurable y posibilidad de adaptar las funciones aplicadas a la prueba de profundidad.

Sería de gran utilidad la inclusión de los parámetros mencionados, ya que esto permitiría la posibilidad de desarrollar un grupo importante de efectos asociados al uso de transparencias, sobras, entre otros.

La implementación desarrollada le permite al usuario incluir variables de tipo uniforme a sus *shaders*, sin embargo, no existe ninguna herramienta que permita registrar variables personalizadas de tipo atributos o uniformes. Una solución sería incorporar un nuevo modal a la interfaz de usuario que permita gestionar (crear, modificar y eliminar) las variables asociadas a un *shader* en particular. Es importante mencionar que los requerimientos necesarios para la implementación de esta funcionalidad del lado servidor fueron previstos, y actualmente el esquema de base de datos y la API REST están preparados para soportarla.

La investigación realizada podría ser la base para el desarrollo de un sistema más complejo que permita el implementar *shaders* en un contexto 2D o post procesamiento. Dicho sistema podría permitir la elección por parte del usuario del tipo de *shader* que se desea codificar y configurar el ambiente de desarrollo en base a la elección. Esto implicaría el desarrollo de módulos adicionales como el manejo de *sprite* y la manipulación de buffers de despliegues propios de OpenGL.

La solución puede ser ajustada para permitir que el usuario guarde el estado actual del sistema; esto es, almacenar la posición, rotación y configuración de despliegue, valores de planos de corte, *shader* actual, configuración del *viewports*, etc. La solución sería capaz de guardar y cargar estos archivos/datos de configuración.

Teniendo en cuenta los resultados obtenidos en las pruebas de rendimiento aplicadas a los shaders, resuelta viable realizar tareas de procesamiento gráfico más exigentes. Sería interesante realizar las adaptaciones necesarias en la solución para realizar despliegues de volúmenes 3D a través de técnicas como *Volume Render*.

Bibliografía

[1] **Williams, James L.** *Learning HTML5 Game Programming.* s.l. : Addison-Wesley, pp. 1-48. 2011.

[2] **Stolarski, Mikołaj.** Merix Studio. WebGL today. Just a hype or future of our industry? [En línea] http://www.merixstudio.com/blog/webgl-hype-future-digital-interactive/.

[3] **Quilez, Inigo y Jeremias., Pol.** Shadertoy. [En línea] https://www.shadertoy.com/.

[4] **Parisi, Tony.** *Programming 3D Applications with HTML5 and WebGL. 3D Animation and Visualization for Web Pages.* s.l. : O'Reilly Media, pp. 39-81. 2014.

[5] **Nobel-Jørgensen, Morten.** Kick JS. [En línea] http://www.kickjs.org/tool/shader_editor/shader_editor.html. 2014.

[6] **Matsuda, Kouichi y Lea, Rodger.** *WebGL Programming Guide. Interactive 3D Graphics Programming with WebGL.* s.l. : Addicon-Wesley, pp. 1-7. 2013.

[7] **Lewis, Paul.** HTML5 Rocks. Canvas Inspection using Chrome DevTools. [En línea] http://www.html5rocks.com/en/tutorials/canvas/inspection. 2014.

[8] **Fulton, Steve y Fulton, Jeff.** *HTML5 Canvas, 2nd Edition. Native Interactivity and Animation for the Web.* s.l. : O'Reilly Media, pp. 1-26. 2011.

[9] **Danchilla, Brian.** *Beginning WebGL for HTML5.* s.l. : Apress, pp. 233-266. 2012.

[10] **Christian.** WebGL Shader Lab. [En línea] http://codedstructure.net/projects/webgl_shader_lab. 2014.

[11] **Cabello, Ricardo , Faye-Lund, Erik y Mackey, Ed .** GLSL Sandbox. [En línea] http://glsl.heroku.com. 2014.

[12] **Mozilla Developer Network.** Firefox Developer Tools. Shader Editor. [En línea] https://developer.mozilla.org/en-US/docs/Tools/Shader_Editor. 2014.

[13] **Fielding, Roy Thomas.** *Architectural Styles and the Design of Network-based* Software *Architectures.* pp. 5-65. 2000.

[14] **Roldós, Guillermo.** *Introducción a WS-REST.* Montevideo, Uruguay, pp. 4-14. 2010.

[15] **Mora, Juan Tahuiton.** *Arquitectura de* software *para aplicaciones Web.* México D.F., pp. 7-66. 2011.

[16] **Google Inc.** Angular JS. [En línea] 2010-2015. https://angularjs.org/.

[17] **Otwell, Taylor.** Laravel. [En línea] 2015. http://laravel.com/.

[18] **Grupo khronos.** OpenGL ES 2.0 for the Web. [En línea] 2015. https://www.khronos.org/webgl/.

[19] **Despoulain, Thibaut.** Shader Editor Shdr. [En línea] 2015. http://shdr.bkcore.com/.

[20] **http://glslsandbox.com/.** Shader Editor GLSL Sandbox. [En línea] 2015. http://glslsandbox.com/.

[21] **Jeremias, Inigo Quilez.** Shader Editor Shadertoy. [En línea] 2015. https://www.shadertoy.com/.

[22] **Cabello, Ricardo.** Three JS. [En línea] 2015. http://threejs.org/.

[23] **Bass, Ben.** Shader Editor Coded Structure. [En línea] 2015. http://codedstructure.net/.