


UNIVERSIDAD CENTRAL DE VENEZUELA
FACULTAD DE CIENCIAS
ESCUELA DE COMPUTACIÓN

DISEÑO Y DESARROLLO DE UN SISTEMA BASE DE GESTION DE CONTENIDOS

Trabajo Especial de grado presentado ante la ilustre
Universidad Central de Venezuela por el
Br. Paulo Cesar Castro Lameda
Para optar por el título de Licenciado en Computación

Tutor: Rafael Rodríguez

Caracas, Mayo 2016

Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación

ACTA DEL VEREDICTO

Quienes suscriben, Miembros del Jurado designado por el Consejo de la Escuela de Computación para examinar el Trabajo Especial de Grado, presentado por el Bachiller Paulo Cesar Castro Lameda de C.I.: 21.411.354, con el título *Diseño y Desarrollo de un Sistema Base de Gestión de Contenidos*, a los fines de cumplir con el requisito legal para optar al título de Licenciado en Computación, dejan constancia lo siguiente:

Leído el trabajo por cada uno de los Miembros del Jurado, se fijó el día 31 de Mayo de 2016, a las 9:30 AM, para que su autor lo defendiera en forma pública, en la Sala 1 de la Escuela de Computación, lo cual se realizo mediante una exposición oral de su contenido, y luego respondió satisfactoriamente a las preguntas formuladas por el Jurado, todo ello conforme a lo dispuesto en la Ley de Universidades y demás normativas vigentes de la Universidad Central de Venezuela. Finalizada la defensa pública del Trabajo Especial de Grado, el jurado decidió aprobarlo con una nota de 19.

En fe de lo cual se levanta la presente acta, en Caracas a los treinta un (31) días del mes de Mayo del año dos mil dieciséis (2016), dejándose también constancia de que actuó como Coordinador del Jurado el Profesor Rafael Rodríguez.

Prof. Rafael Rodríguez
(Tutor)

Prof. Mercy Ospina
(Jurado Principal)

Prof. Néstor Méndez
(Jurado Principal)

Resumen

En el siguiente tomo se detallan las actividades realizadas en el Trabajo Especial de Grado (TEG) cuyo objetivo fue el Desarrollo un Sistema Manejador de contenido (CMS) enfocado en la accesibilidad del producto y el manejo de versiones, con un conjunto de funcionalidades básicas para la generación de interfaces web que esté construido sobre tecnologías escalables, robustas y de calidad que faciliten su extensión y crecimiento. El desarrollo no implico solamente la implementación de dicho CMS, sino también el diseño de toda su arquitectura y la especificación de cómo se relacionan las tecnologías que la componen. Para esto fue necesaria la realización de una etapa de investigación de tecnologías, arquitecturas y enfoques que permitieran alcanzar los objetivos planteados de la manera más óptima. Luego se escogió una metodología a ser aplicada para comenzar con el desarrollo del Sistema Manejador de contenidos, dicho desarrollo consistió en el diseño e implementación de dos módulos principales, la aplicación de entrega de contenido y la aplicación de manejo de contenido. Una vez finalizada la implementación de dichos módulos, se mostraron los resultados del desarrollo a través de la elaboración de una página web sumamente sencilla, para mostrar el uso de las funcionalidades básicas pero cruciales que fueron desarrolladas.

Palabras Clave: Aplicación web, *Frameworks* de desarrollo web, Aplicaciones de escritorio, Sistema de gestión de Contenido, Aplicación de Entrega de Contenido, Aplicación de Manejo de Contenido, Manejo de Versiones, *single page aplication*.

A mi Madre Auristela,
Quien me lo ha dado todo
Y es mi más grande ejemplo a seguir
Para llegar a ser una gran persona.

Agradecimientos

A mi madre Auristela mi más grande ejemplo.

A mis hermanos Diego y Nelson que siempre me han apoyado.

A los profesores de la Facultad de Ciencias por su gran dedicación y perseverancia.

A mi tutor Rafael Rodríguez, Gracias a su apoyo a lo largo de este trabajo.

A la profesora Mercy Ospina, por sus consejos y ayuda para el debido desarrollo de este proyecto

A mis tías Alejandra y Angelina, mi prima María y mi abuela Magdalena, por siempre estar pendientes de mí.

Y finalmente a todas las personas que me han retado a ser mejor ser humano y profesional.

Gracias.

Índice General

Índice de Figuras

Índice de Tablas

Introducción

En los últimos años el mundo de las aplicaciones y páginas web ha crecido y evolucionado de manera masiva, llegando a formar parte importante de varios aspectos de la vida del ser humano ya sea en un ámbito social, personal o de negocio. Se podría decir que ahora las personas no cuentan solo con su identidad real dentro de la sociedad, sino también con una identidad digital ligada a esta nueva era de la información. Tanto para las personas como para las empresas esta identidad digital es de gran importancia ya que forma parte de lo que los describe.

Las redes sociales y blogs les han permitido a las personas mostrar tanto aspectos de su vida personal como de su vida profesional, de múltiples maneras. Mientras que los manejadores de contenido han facilitado la creación de aplicaciones o páginas web más personalizadas y con un propósito más específico.

No obstante, existe un problema recurrente en la sociedad actual. Aunque existen muchas formas sencillas y completas de construir una imagen digital, ya sea a través de redes sociales, Blogs o manejadores de contenido, éstas te restringen a un conjunto de características y tecnologías que finalmente terminan limitando el crecimiento o adaptabilidad de manera sencilla, obligándote a empezar de cero en caso de que se sobrepase su alcance.

Específicamente los manejadores de contenido suelen ser muy sencillos de usar, permitiendo crear páginas o aplicaciones web de manera rápida y simple, pero su enfoque en la usabilidad ha dejado a un lado la posibilidad de usar nuevas arquitecturas que han ido surgiendo en el mundo de las aplicaciones web, las cuales satisfacen nuevas necesidades de las imágenes digitales.

Para la persona común o las pequeñas empresas este límite puede convertirse en un problema cuando se le presentan oportunidades de crecimiento. Esto las obliga a invertir en el desarrollo de páginas o aplicaciones web que sean capaces de cumplir no solo con sus nuevos requerimientos, sino también con los antiguos, condicionando el crecimiento de su identidad digital a su capacidad de inversión.

Es por esto que el Presente trabajo especial de grado (TEG) propone el desarrollo de un nuevo Sistema Manejador de Contenidos que provea soluciones con arquitecturas más propicias para satisfacer algunas de las nuevas necesidades de hoy en día.

Dicho trabajo se estructura en capítulos descritos a continuación:

- Capítulo 1: Se plantea el problema y se describen brevemente los manejadores de contenido particularmente famosos actualmente, junto con los objetivos generales y específicos de esta investigación y su justificación.
- Capítulo 2: Se abordaran los distintos conceptos y tecnologías consideradas relevantes para el problema planteado en el capítulo anterior.
- Capítulo 3: Se define la metodología a usarse junto con las distintas ventajas que ofrece para el desarrollo debido y controlado de la solución planteada.
- Capítulo 4: Son descritas las iteraciones del desarrollo en cada uno de sus pasos, señalando todas ocurrencias y eventualidades importantes para luego dar una conclusión sobre cada iteración.
- Capítulo 5: Se mencionan las distintas conclusiones de este trabajo especial de grado y las recomendaciones asociadas a estas.

CAPITULO I: Planteamiento de Investigación

1.1 Planteamiento del problema

En los últimos años muchas personas y/o empresas han creado páginas web, estas pueden ser de cualquier tipo, informativas, académicas, comerciales o sociales, entre otras. Ambas hacen uso de ellas para difundir una imagen propia, que va cambiando y creciendo a través del tiempo. Pero una vez esta imagen cambia o crece demasiado, se vuelve difícil ser capaz de representarla o mantenerla de manera rápida y sencilla.

Los manejadores de contenido son herramientas de alto nivel que han facilitado la creación y modificación de páginas web hasta tal punto que ya no se requiere de profesionales cuya formación incluye el desarrollo de páginas web para el desarrollo de una.

Sin embargo, varias de estas herramientas suelen generar soluciones construidas sobre tecnologías y arquitecturas que se han ido quedando atrás con respecto a otras más nuevas, llegando a limitar las características del producto generado utilizándolas. Algunas de estas arquitecturas y tecnologías más nuevas han demostrado ser más propicias para satisfacer nuevas necesidades que han ido surgiendo en el mundo de las aplicaciones web.

Se estima que un 25% de toda la web está construida con manejadores de contenido. Esto da aun mayor énfasis a la necesidad de hacer que estas herramientas puedan ofrecer la mayor cantidad de alternativas para la generación de sus soluciones. Para así poder garantizar un buen rendimiento y calidad en estas.

Por lo anteriormente mencionado, el continuo crecimiento que sigue mostrando la web y como cada vez las personas y organizaciones muestran una mayor necesidad de dar un buen mantenimiento a su imagen digital, el autor plantea la siguiente interrogante: ¿Es

posible crear un manejador de contenido que sea capaz de generar soluciones con una arquitectura que se ajuste a las nuevas necesidades de hoy en día?

Al plantearse esta interrogante surgen varios requerimientos que la herramienta debe cumplir para que pueda generar soluciones que mantengan un buen nivel de calidad:

- Altamente modular, con cada módulo teniendo un alto nivel de granularidad.
- Proveer una arquitectura orientada a servicios en sus soluciones que facilite la posibilidad de acceder a servicios y/o funcionalidades de terceros.
- Construir dichas soluciones sobre tecnologías de punta que se distingan por su buen rendimiento, escalabilidad y calidad.

Además de todas las funcionalidades que todo Manejador de contenido debe poseer:

- Proveer plantillas de código fácilmente modificables.
- Editor de código integrado.
- Una interfaz usable y accesible.
- Estilos predeterminados y facilidad para incluir nuevos.
- Herramienta para diseño de interfaces.
- Manejo de versiones de distintos proyectos.

Este Trabajo Especial de Grado (TEG) tiene como finalidad el diseño y desarrollo de un manejador de contenido base con un conjunto de funcionalidades básicas que en un futuro la permita cumplir con las características anteriormente mencionadas, apoyándose en la metodología SCRUM en cada paso de su desarrollo para cumplir con los objetivos y el alcance planteados.

1.2 Antecedentes

Dentro los antecedentes de esta investigación vale la pena mencionar varias de las ventajas y desventajas de algunos de los manejadores de contenido más utilizados en la actualidad, para así resaltar la importancia del desarrollo de un manejador de contenidos que ofrezca buenas soluciones donde estas otras herramientas no brillan tanto.

1.2.1 Wordpress

Es un manejador de contenidos enfocado a la creación de cualquier tipo de sitio web. Fue desarrollado en el lenguaje PHP para entornos que ejecuten MySQL y Apache, bajo licencia GPL y es software libre. Su fundador es Matt Mullenweg. WordPress fue creado a partir del desaparecido b2/cafelog y se ha convertido en uno de los CMS más populares. Las causas de su enorme crecimiento son, entre otras, su licencia y su facilidad de uso [1].

Otro punto a considerar sobre su éxito y extensión es la enorme comunidad de desarrolladores y diseñadores, encargados de programarlo en su núcleo o creando complementos (llamados plugins) y plantillas (llamados temas) para la comunidad. En febrero de 2015 era usado por el 23,4% de todos los sitios existentes en Internet basados en gestores de contenido [2].

Las críticas de WordPress se han centrado varias veces alrededor de su seguridad, muchos problemas de seguridad no han sido resueltos en el software, particularmente entre 2007 y 2008 las inyecciones SQL presentaron problemas graves [3].

Otro problema que presenta Wordpress es su bajo nivel de rendimiento en relación a las tecnologías sobre las que está construido, aunque existen varios métodos para mejorar su desempeño, este aun demuestra no ser óptimo en comparación con otras tecnologías. Al observar la figura 1 podemos observar que su rendimiento en relación a estas es muy pobre.

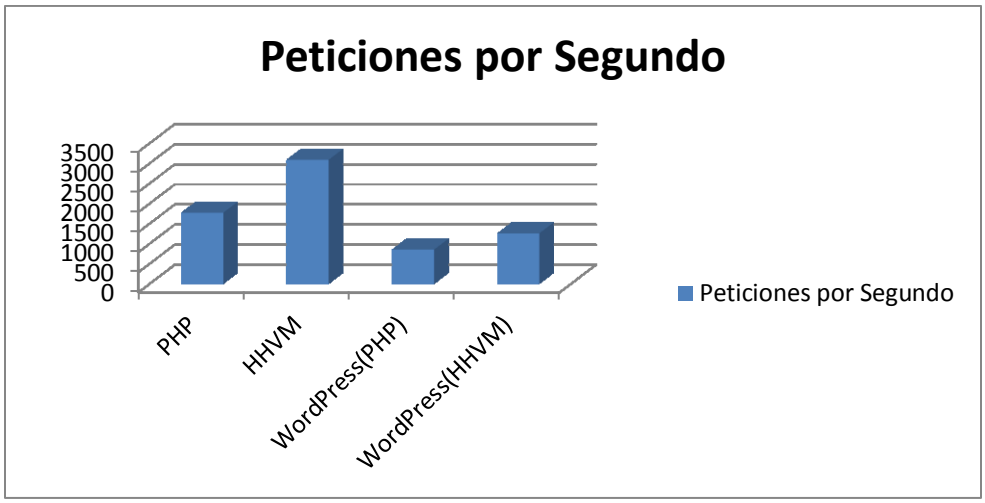

Figura 1. HostingAdvice (2015). Simple HTTP Requests, [Figura]. Recuperado de: [4]

1.2.2 Drupal

Drupal es el tercer sistema de gestión de contenidos más popular disponible en la actualidad. Es un programa de código totalmente abierto y se distribuye bajo los términos de la Licencia Pública General de GNU (GPL) [5]. La plataforma de Drupal es muy potente, y requiere menos recursos que el de WordPress. Drupal se puede configurar para cualquier cosa, desde un simple blog a un portal de contenidos utilizados por las grandes corporaciones. Algunas de las características más significativas de Drupal incluyen:

- Técnicamente avanzado: este requiere más conocimientos técnicos que la mayoría de los sistemas de gestión de contenidos. Utiliza mucha menor cantidad de recursos del sistema que otros CMS como WordPress, por lo que la gente no tendrá que preocuparse acerca de la actualización a una opción de alojamiento más costosa de manera tan repentina.

- Rendimiento mejorado: páginas de Drupal suelen cargar más rápidamente, y tienen tiempos de respuesta más rápidos que las fabricadas con WordPress u otros CMS (ver figura 2). Por supuesto, a medida que se agregan plug-ins y se hacen otros cambios, esto puede cambiar rápidamente.

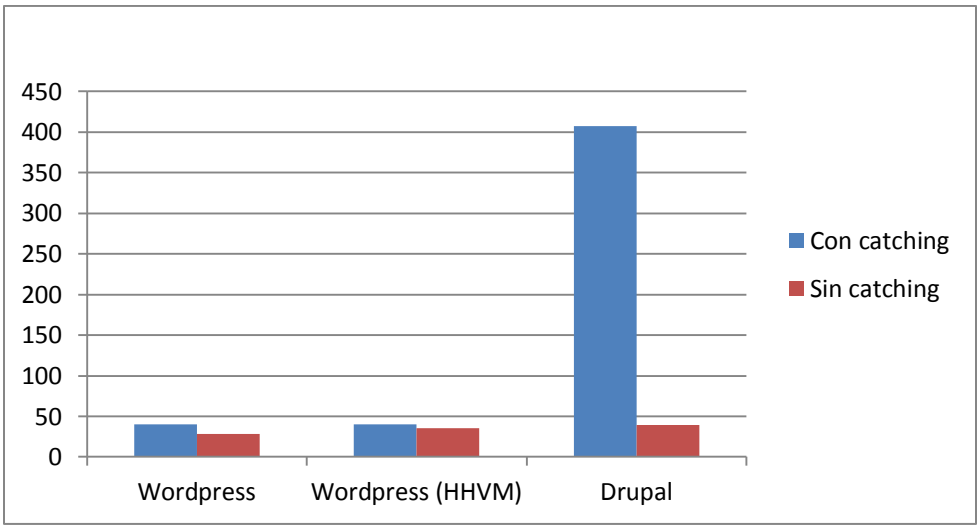

Figura 2. Geerling, J. (2015). BenchMarking PHP7 vs HHVM – Drupal and Wordpress, [Figura]. Recuperado de: [6]

- Personalizable: Drupal es fácil de personalizar con muchos plug-ins diferentes, temas y otras opciones configurables. Para aquellos con conocimientos de programación suficiente, es posible editar los que los archivos root del programa, por lo que es el más flexible que otros CMS.

- Bases de Datos: Además de ofrecer el uso de distintas Bases de Datos relacionales, a partir de su versión 7, Drupal ofrece a MongoDB como opción no relacional.

Sin embargo a pesar de estas distintas ventajas de Drupal, este sigue ofreciendo una arquitectura cliente-servidor clásica en sus soluciones, además de que requiere conocimientos de programación en todas la etapas del desarrollo de una aplicación, ya que es un CMS más orientado a los programadores.

1.2.3 Joomla

Joomla es planteado a menudo como el punto medio entre WordPress y Drupal. Se trata de un CMS bajo la licencia GPL, que pueden funcionar sin problemas en la mayoría de los servidores web sin ningún problema y con mayor escalabilidad que Wordpress [5]. Que no requiere el mismo nivel de experiencia técnica para funcionar como Drupal, pero todavía ofrece muchas de las características adicionales. Como Drupal y WordPress, Joomla tiene una gran cantidad de plug-ins y temas disponibles para elegir, para que pueda personalizar su sitio para mirar y funcionar en cualquier forma que desee. Otras razones que la gente elige Joomla incluyen:

- Redes sociales: Esta es quizás la ventaja más grande de Joomla. De los tres, Joomla hace que sea más fácil de crear redes sociales. Las redes sociales pueden ser un activo de gran alcance para muchos sitios, y con Joomla, usted puede tener uno en marcha y funcionando de forma rápida y fácil.

- Técnicamente Accesible: Joomla no requiere conocimientos avanzados de programación para la mayoría de sus funcionalidades, aunque para algunas en específica puede que se requiera de algún programador con experiencia.

1.2.4 Comparando Wordpress, Drupal y Joomla

A la hora de comparar estos 3 CMS existen varios criterios a tomar en cuenta, pero principalmente dentro del objetivo de la investigación nos interesa la escalabilidad de sus soluciones, es decir, la capacidad para manejar el incremento continuo de trabajo con fluidez y sin decrementos graves de desempeño, dando a entender que dichas soluciones tienen la capacidad para crecer manteniendo la calidad en todos sus servicios.

Tomando como referencia un estudio realizado por Yotta una compañía dedicada al desarrollo y optimización de aplicaciones web [7], el figura 3 podemos observar los siguientes resultados:

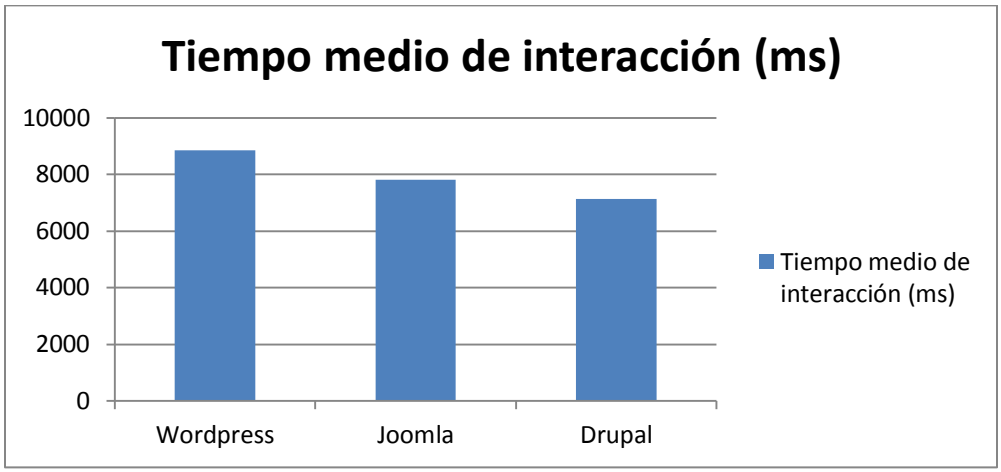

Figura 3. Yotta, Tiempo medio de Interacción, [Figura]. Recuperado de: [8]

Interpretando el tiempo medio de interacción como el promedio de tiempo de respuesta entre cada solicitud HTTP realizada por la aplicación web, podemos ver que Drupal en general muestra un mejor desempeño y que Wordpress es claramente el más lento. Aunque la diferencia es tan solo de 1segundo, hay que tomar en cuenta que el estudio fue realizado sobre más de 15mil sitios web de 8 distintos CMS [8], entre los cuales están los 3 que ahora comparamos.

Otro ángulo a tener en cuenta es el rendimiento del backend (ver figura 4). A pesar de que el principal cuello de botella para el rendimiento se ha desplazado al contenido de la página (el frontend), la entrega de los distintos elementos HTML, CSS y JavaScript sigue siendo un obstáculo que debe ser tratado por todos los propietarios de sitio. Después de todo, nada en la página puede cargar (ser mostrado) hasta que estos elementos han sido entregados.

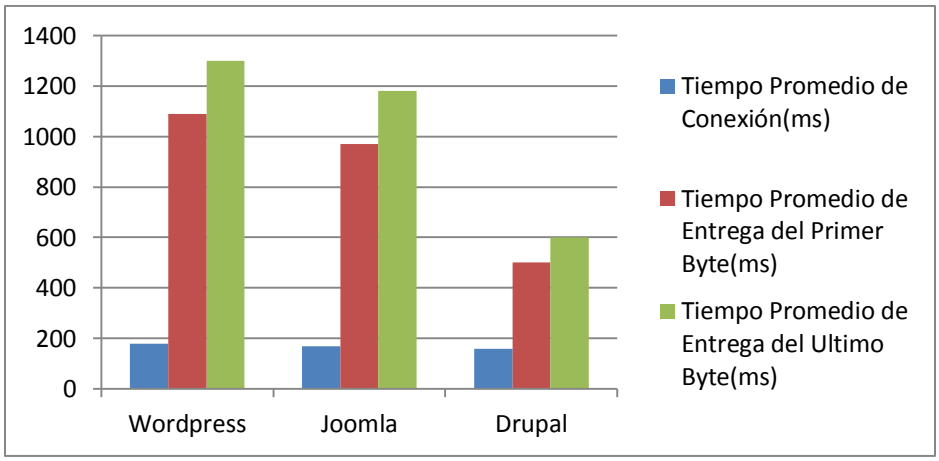

Figura 4. Yotta. Average Backend Metrics. [Figura]. Recuperado de: [8]

En la figura 4 podemos observar que tanto Joomla como Wordpress presentan cierto retardo en la entrega de estos recursos en comparación a Drupal, apoyando lo anteriormente mencionado con respecto a Drupal siendo capaz de generar soluciones con mejor rendimiento.

1.3 Objetivo General

Desarrollar un Sistema Manejador de contenido (CMS) enfocado en la accesibilidad del producto y el manejo de versiones, con un conjunto de funcionalidades básicas para la generación de interfaces web que esté construido sobre tecnologías escalables, robustas y de calidad que faciliten su extensión y crecimiento.

1.4 Objetivo Especifico

Dentro del objetivo general del desarrollo de un manejador de contenido, se identificaron los siguientes objetivos específicos:

- Captar los Distintos Requerimientos
- Diseñar la solución
    - Definir alcance de la solución

- Analizar tecnologías a utilizar:
  - Para la accesibilidad del código y manejo de versiones
  - Para el desarrollo de Web API
  - Para el desarrollo de la Aplicación de manejo de contenido
- Elaboración de distintos diagramas:
  - Diagrama de Despliegue
  - Diagrama de Objetos del Dominio
  - Diagrama de Casos de Uso

- Desarrollar un servicio web o ***back-end*** específico para el manejo de proyectos y usuarios de la herramienta
  - Implementar un API REST que contenga toda la lógica necesaria para la administración de proyectos generados con la herramienta
  - Manipulación de distintas versiones de archivos y repositorios de código
  - Manejo de usuarios con distintos niveles de privilegio
  - Elaboración de la documentación concerniente al API
- Desarrollar una aplicación de escritorio o ***front-end,*** específica para la creación y edición de proyectos pertenecientes a los usuarios de la herramienta.
  - Diseño de interfaz para la creación y modificación de proyectos
  - Módulo de diseño de interfaces
  - Módulo de manejo de versiones de proyecto
- Implementar la herramienta en un ambiente de producción
- Desarrollar una página web con el objetivo de funcionar como caso de estudio de las funcionalidades de la plataforma

1.5 Justificación

En el planteamiento del problema se hizo mención a como las personas y las organizaciones de hoy en día cuentan con una identidad digital que es de suma importancia en sus vidas, ya sea por razones personales, académicas o profesionales. Es natural que las personas busquen la forma más sencilla de construir y mantener esta identidad, una posible prueba de esto es que más del 25% de las aplicaciones y páginas web, están hechas a través manejadores de contenido. Pero es inevitable que surjan dudas como: ¿Facilitar el trabajo necesariamente implica limitar las características que pueda tener tu producto? Vale la pena tomar en cuenta la alta probabilidad de que una cantidad representativa de los portales realizados con estas herramientas necesiten seguir creciendo ya sea a través de la extensión de sus funcionalidades o la mejora de su desempeño, pero las soluciones generadas a través de las herramientas más populares actualmente no permiten hacer esto de manera sencilla.

Por lo tanto es necesario e importante proveer de herramientas y mecanismos que contribuyan a la facilidad de crecimiento de las identidades digitales de las personas y organizaciones, garantizándoles no solo la existencia de su identidad digital, sino también el futuro crecimiento y continua mejora.

CAPITULO II: Marco Conceptual

2.1 Conceptos

A continuación se plantearan algunos conceptos y paradigmas relevantes para el desarrollo de la investigación planteada.

2.1.1 Sistemas Manejadores de Contenido (CMS)

Un sistema de gestión de contenidos (CMS) es un sistema utilizado para administrar el contenido de un sitio Web. Por lo general, un CMS se compone de dos elementos: la aplicación de gestión de contenidos (CMA) y la aplicación de entrega de contenido (CDA) [9]. La aplicación de gestión de contenidos permite al personal o autor encargado de la página, que no conozca HTML, gestionar la creación, modificación y eliminación de contenido de un sitio web sin necesidad de algún profesional del área. Mientras que la aplicación de entrega de contenido usa y compila esa información para actualizar el sitio Web. Las características de un sistema CMS varían, pero la mayoría incluyen la edición basada en la Web, gestión de formatos, control de revisión, indexación, búsqueda y recuperación.

La función de publicación basada en Web permite a las personas a utilizar una plantilla o un conjunto de plantillas, así como asistentes y otras herramientas para crear o modificar el contenido Web.

Se podría decir que las aplicaciones creadas con un manejador de contenido siguen el mismo siclo de vida que otras aplicaciones (ver Figura 5), pero las distintas etapas de este siclo se ven simplificadas por la facilidad de uso de la herramienta CMS.

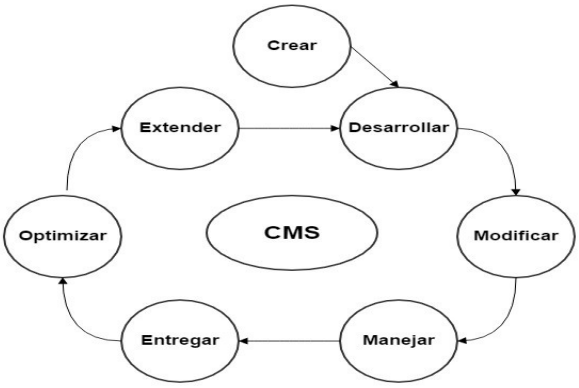

Figura 5. Ciclo de vida de Aplicaciones, "Elaboración Propia"

Dentro las características generales y posibles ventajas o limitaciones que se observan al hacer una aplicación web con un CMS tenemos:

- Arquitectura clásica cliente-servidor, donde no existe una diferenciación real entre el *frontend* y el *backend* y donde no se suele implementar una arquitectura como la orientada a servicios.
- El código generado es sumamente genérico para abarcar la mayor cantidad posible de casos lo que lo hace difícil de optimizar para que la aplicación mantenga un buen rendimiento.
- Desarrollo sumamente rápido y sencillo gracias a una gran cantidad de temas y funcionalidades predeterminadas en la herramienta o desarrolladas por la comunidad de la herramienta en cuestión.

La arquitectura clásica cliente-servidor en las aplicaciones web generadas por los CMS de hoy en día suele perder rendimiento a medida que las aplicaciones van creciendo, ya que somete al servidor a una carga de procesamiento por la renderización de código HTML, CSS y JavaScript de cada vista perteneciente a la aplicación web. Mientras que nuevos paradigmas para la construcción de aplicaciones web como las Single Page Application (SPA) junto con una arquitectura orientada a servicio no solo ofrece mayor escalabilidad sino también mayor extensibilidad.

2.1.2 Aplicaciones Web

En las aplicaciones web suelen distinguirse tres niveles: el nivel superior que interacciones con el usuario, normalmente a través de un navegador, el nivel inferior que proporciona los datos y el nivel intermedio que procesa los datos.

Una aplicación web es un tipo especial de aplicación cliente/servidor, donde tanto el cliente (el navegador web) como el servidor y el protocolo (HTTP) mediante el que se comunican están estandarizados y no han de ser creados por el programador de aplicaciones (ver Figura 6).

El protocolo HTTP forma parte de la familia de protocolos de comunicaciones *Transmission Control Protocol/Internet Protocol* (TCP/IP), que son los empleados en internet. Estos protocolos permiten la conexión de sistemas heterogéneos, lo que facilita el intercambio de información entre distintos ordenadores [10].

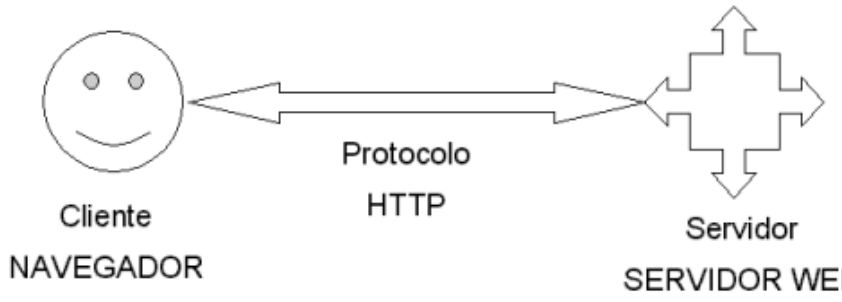

Figura 6. Mora, L. (2016). Esquema básico de una aplicación web. [Figura] Recuperado de: **[10]**

Entonces podríamos decir que una aplicación web es un conjunto de herramientas utilizadas por el usuario que interactúan entre sí con diversos recursos en un servidor web a través de una red de internet o intranet desde un navegador web. Esta tecnología también abarca la manipulación de bases de datos para la realización de consultas y otras operaciones. Las aplicaciones web presentan una fuerte ventaja en cuanto a simplicidad de actualización o compatibilidad entre múltiples plataformas, ya que no se requiere crear distintas versiones según sistemas operativos diversos sino que estas pueden ser accedidas desde cualquier navegador, además de que no consumen espacio en disco duro y requieren hardware poco costoso.

2.1.3 Arquitectura Cliente-Servidor

Desde el punto de vista funcional, se puede definir la computación Cliente/Servidor como una arquitectura distribuida que permite a los usuarios finales obtener acceso a la información en forma transparente aún en entornos multiplataforma [11].

En el modelo cliente servidor, el cliente envía un mensaje solicitando un determinado servicio a un servidor (hace una petición), y este envía uno o varios mensajes con la respuesta (provee el servicio). En un sistema distribuido cada máquina puede cumplir el rol de servidor para algunas tareas y el rol de cliente para otras.

La idea es tratar a una computadora como un instrumento, que por sí sola pueda realizar muchas tareas, pero con la consideración de que realice aquellas que son más adecuadas a sus características [11]. Si esto se aplica tanto a clientes como servidores se entiende que la forma más estándar de aplicación y uso de sistemas Cliente/Servidor es mediante la explotación de las PC's a través de interfaces gráficas de usuario; mientras que la administración de datos y su seguridad e integridad se deja a cargo de computadoras centrales tipo mainframe. Usualmente la mayoría del trabajo pesado se hace en el proceso llamado servidor y el o 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 la finalidad 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 grandemente [11].

2.1.3.1 El Cliente Web

El cliente web es un programa con el que interactúa el usuario para solicitar a un servidor web el envío de los recursos que desea obtener mediante llamadas HTTP, se le conoce el término *front-end***.** Este se encarga de mostrar y administrar la interfaz al usuario**.** Suele estar formada por código HTML (*HyperText Markup Lenguaje*) y CSS (C*ascading Style Sheets*) que forma la página web más algo de código ejecutable realizado en lenguaje script del navegador (*Javascript* o *VBScript*). Por lo tanto, la misión del cliente web es interpretar las paginas HTML y los diferentes recursos que contienen (imágenes, sonidos, etc.), además de realizar llamadas HTTP asíncronas (AJAX) para seguir actualizando el contenido que me muestra la usuario [10]**.**

El Cliente normalmente 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 aplicación y hacer validaciones locales.
- Generar requerimientos de bases de datos.
- Recibir resultados del servidor.
- Formatear resultados.

2.1.3.2 El Servidor

Es el proceso encargado de atender a múltiples clientes que hacen peticiones de algún recurso administrado por él. Al proceso servidor se le conoce con el término *back-end* [11].

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.
- Formatear datos para trasmitirlos a los clientes.
- Procesar la lógica de la aplicación y realizar validaciones a nivel de bases de datos.

2.1.4 Modelo-Vista-Controlador

Se desarrolló la metáfora Modelo-Vista-Controlador y su paradigma, para la estructuración adecuada de aplicaciones y componentes de aplicaciones interactivas (ver Figura 7). Los modelos son aquellos componentes de la aplicación del sistema que realmente hacen el trabajo. Se mantienen desligados de las vistas, que muestran aspectos de los modelos. Los controladores se utilizan para enviar mensajes al modelo y proporcionar la interfaz entre el modelo con sus vistas asociadas y los dispositivos de interfaz de usuario interactivos (por ejemplo, teclado, ratón). Cada vista puede ser considerada como estrechamente relacionada con un controlador, cada una con exactamente un modelo, mientras que un modelo puede tener muchos pares vista / controlador.

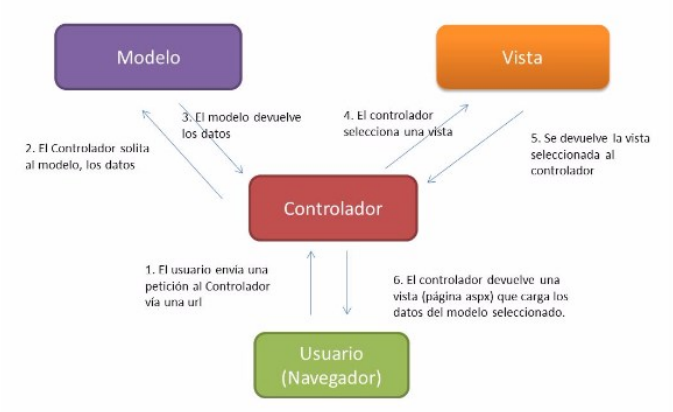

Figura 7. MiBlogTecnico. (2016). Modelo-Vista-Controlador. [Figura]. Recuperado de: https://miblogtecnico.wordpress.com/tag/asp-net-mvc-2/

- **Modelo:** El modelo de una aplicación es la implementación de la estructura central de la aplicación [12]. Es la representación de la información con la cual el sistema opera, por lo tanto gestiona todos los accesos a dicha información, tanto consultas como actualizaciones, implementando también los privilegios de acceso que se hayan descrito en las especificaciones de la aplicación (lógica de negocio). Esto puede ser tan simple como un número entero (como el modelo de un contador) o cadena (como el modelo de un editor de texto), o puede ser un objeto complejo o implementación de algún esquema relacional a nivel de base de datos [3]. Envía a la 'vista' aquella parte de la información que en cada momento se le solicita para que sea mostrada (típicamente a un usuario). Las peticiones de acceso o manipulación de información llegan al 'modelo' a través del 'controlador'.

- **Controlador:** Responde a eventos (usualmente acciones del usuario) e invoca peticiones al 'modelo' cuando se hace alguna solicitud sobre la información (por ejemplo, editar un documento o un registro en una base de datos). También puede enviar comandos a su 'vista' asociada si se solicita un cambio en la forma en que se presenta el 'modelo'.

- **Vista:** Se encarga de presentar el 'modelo' en un formato adecuado para interactuar con el usuario y envía mensajes a través su controlador pareja para realizar peticiones al modelo.

2.1.5 Mapeo Objeto-Documento (ODM)

El Mapeo Objeto-Documento (Object-Document mapping, en su siglas en inglés) es una técnica de programación muy utilizada para aplicaciones que utilizan bases de datos documentales, sirve para conectar objetos propios de una aplicación, con los documentos de un sistema manejador de base de datos documental. Se diferencia de los ORM (Object-Relation mapping en su siglas en inglés) ya que este no busca solo mapear la relaciones entre objetos sino también aplicar restricciones e implementar un esquema sobre estos,

ya que la mayoría de las bases de datos documentales, no te permiten declarar un esquema o modelo, es necesario implementar las reglas de negocio a nivel de aplicación.

Un ODM bien formado debería ser capaz de:

- Representar los modelos y sus datos.
- Representar asociaciones entre dichos modelos.
- Representar herencias jerárquicas a través de los modelos.
- Validar los modelos antes que estos sean persistidos en una base de datos.
- Permitir operaciones de base de datos de una manera orientada a objetos.

2.1.6 Servicios Web

Existen múltiples definiciones sobre lo que son los Servicios Web, lo que muestra su complejidad a la hora de dar una adecuada definición que englobe todo lo que son e implican. Una posible sería hablar de ellos como un conjunto de aplicaciones o de tecnologías con capacidad para inter-operar en la Web [13]. Estas aplicaciones o tecnologías intercambian datos entre sí con el objetivo de ofrecer unos servicios (ver Figura 8). Los proveedores ofrecen sus servicios como procedimientos remotos y los usuarios solicitan un servicio llamando a estos procedimientos a través de la Web.

Estos servicios proporcionan mecanismos de comunicación estándares entre diferentes aplicaciones, que interactúan entre sí para presentar información dinámica al usuario. Para proporcionar interoperabilidad y extensibilidad entre estas aplicaciones, y que al mismo tiempo sea posible su combinación para realizar operaciones complejas, es necesaria una arquitectura de referencia estándar.

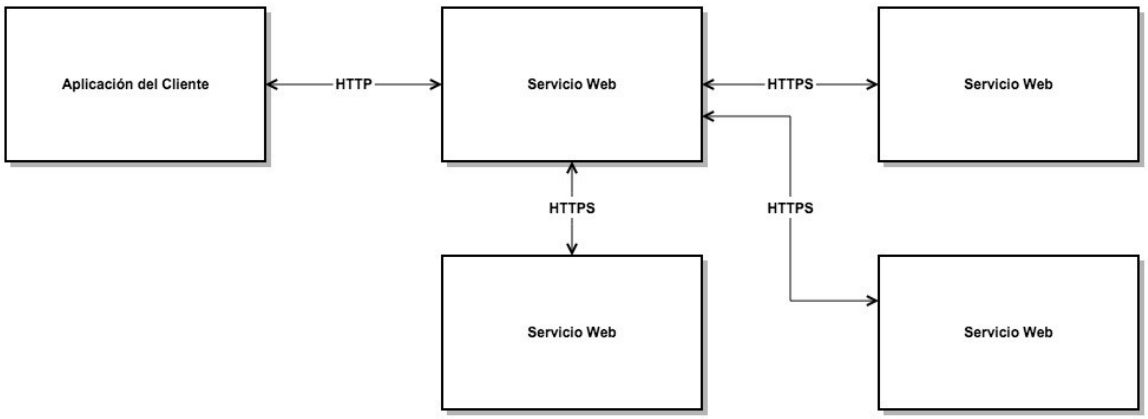

Figura 8. Ejemplo de Servicio Web. "Elaboración Propia"

En la imagen (Ver Figura 8) podemos apreciar un sencillo ejemplo de cómo servicios web pueden interactuar entre sí. En muchas ocasiones estos servicios web no interactúan directamente con aplicaciones de *front-end*, sino con otros servicios web. El alcance de su uso va desde realización de pagos o transacciones monetarias, hasta solicitud credenciales de otras aplicaciones.

2.1.7 Arquitectura Orientada a Servicios (SOA)

Definimos SOA como un estilo arquitectónico donde los sistemas se componen de los usuarios de servicios y proveedores de servicios. Un estilo arquitectónico que define un vocabulario de los componentes y tipos de conectores, y las limitaciones en la forma en que se pueden combinar [14]. Para SOA, los tipos de componentes básicos son usuario del servicio y el proveedor de servicio. Tipos de componentes auxiliares, tales como el *Enterprise Service Bus* (ESB) y el directorio de servicios, pueden ser utilizados.

Los tipos de conectores de SOA incluyen llamadas síncronas y asíncronas usando el protocolo HTTP en arquitecturas RESTful, y la infraestructura de mensajería.

Muchas de las propiedades se pueden asignar a estos tipos de componentes y conectores, pero por lo general son específicos para cada tecnología de implementación. Algunas de las restricciones que se aplican a la arquitectura SOA son:

- Los usuarios de Servicio envían peticiones a los Proveedores de Servicio.
- Un proveedor de servicio también puede ser un usuario de servicio.
- Usuarios de servicio pueden descubrir dinámicamente nuevos servicios a través del directorio de servicios.
- Un ESB puede mediar la interacción entre usuarios de servicio y proveedores de servicio.

Vale la pena mencionar que aunque se ha escrito mucho acerca de SOA y servicios web, todavía hay cierta confusión entre estos dos términos entre los desarrolladores de software. SOA es un estilo de arquitectura, mientras que los servicios Web es una tecnología que se puede utilizar para implementar SOA [14].

2.1.8 Interfaz de Programación de Aplicaciones (API)

El concepto hace referencia a los procesos, las funciones y los métodos que brinda una determinada biblioteca de programación a modo de capa de abstracción para que sea empleada por otro programa informático [15].

Puede entenderse a la API como una interfaz que indica a las aplicaciones cómo pueden **m**antener una comunicación entre sí. Estas reglas permiten que los distintos programas mantengan interacciones.

Otra manera de comprender qué es una API se vincula al suministro de funciones que tienen un uso extendido. De este modo, un programador puede recurrir a la funcionalidad de una API y así evitar iniciar la tarea de programación desde cero. Gracias su flexibilidad, las API ayudan a evitar pasos ya que el programador acude a ellas cuando desarrolla un nuevo programa, reutilizando códigos cuyo funcionamiento ya está probado.

Una API sirve para establecer una comunicación con una base de datos, un sistema operativo o un protocolo de comunicaciones, por citar algunas posibilidades. Incluso las redes sociales emplean distintas API.

Una de las claves del funcionamiento de las API es la facilidad de integración. Estas herramientas tienen que resultar simples de integrar a otro software para que las comunicaciones puedan desarrollarse con éxito. De igual manera, sus actualizaciones no deberían generar conflictos para que su labor siga siendo óptima.

2.1.8.1 API REST

Una REST API es una API, o librería de funciones, a la que se accede por el protocolo HTTP. Por tanto, se accede a través de direcciones web o URLs en las que enviamos los datos de nuestra consulta. Como respuesta a la consulta sobre el REST API se obtienen datos en diferentes formatos, como pueden ser texto plano, XML, JSON entre otros.

Otra esencia de los REST API es el debido uso de los métodos HTTP y la formación de los URL con respecto a estos. Es decir que un mismo URL sea el punto de acceso a distintas funcionalidades para manipular un mismo recurso según el método HTTP usado:

- GET obtiene el recurso
- POST crea un recurso
- PUT modifica un recurso y lo crea si este no existe
- PATCH modifica un recurso
- DELETE borra el recurso (ya sea de manera permanente o temporal)

2.1.9 Aplicaciones de una Sola Página (SPA)

Una Aplicación de una Sola Página (Single-page application, en inglés), es una aplicación web que encaja en una sola página web dando una experiencia al usuario similar a la de una aplicación de escritorio. Al igual que cualquier página web, es necesario código HTML,

Javascript y CSS, solo que este es obtenido en su mayoría en la primera carga de una aplicación SPA. Esta primera carga usualmente recibe el nombre de Bootstraping. Una vez que la primera carga es completada y el cliente web renderiza la página, no hay necesidad de recargar nuevamente y, los futuros cambios de contenidos se realizarán generalmente con peticiones AJAX o WebSockets.

Al final la interacción de un Web API con una aplicación en el lado del cliente SPA, sería de la siguiente manera (ver Figura 9):

- Se realiza la primera carga de archivos HTML, CSS y Javascript, que usualmente se le conoce como Bootstraping.
- La lógica de la SPA estará en los archivos Javascripts, ya alojados en el cliente web.
- La actualización posterior de componentes de la aplicación web se realizaran mediante peticiones HTTP en segundo plano, que pueden ser realizadas con la técnica AJAX.
- El Web API al recibir peticiones AJAX, responderá con lenguajes de comunicación como XML o JSON.
- Una vez que el cliente web reciba las respuestas en JSON o XML, la lógica de la SPA se deberá de encargar de actualizar de manera dinámica la Vista al usuario.

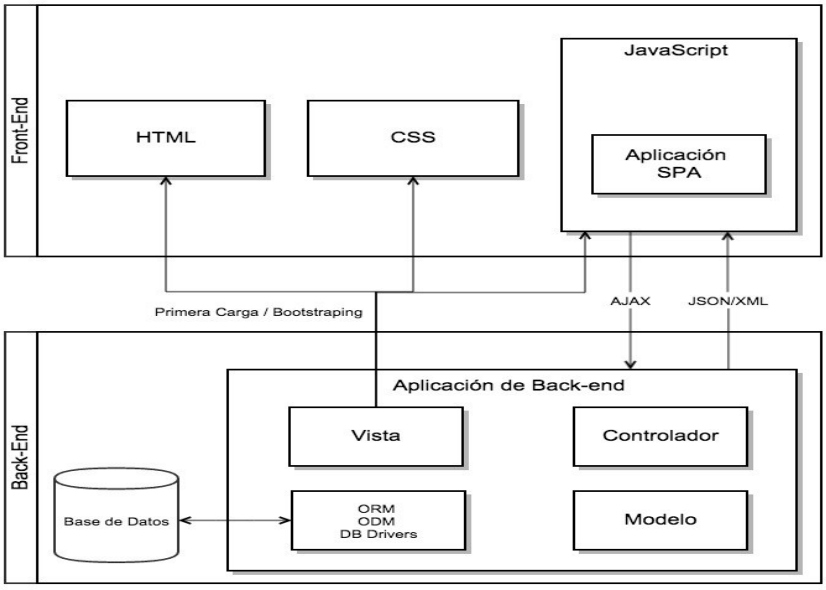

Figura 9. Estructura SPA-API, "Elaboración Propia"

Si comparamos que tan eficiente es la utilización de una SPA contra la utilización de un esquema cliente-servidor clásico en cual la gran mayoría de respuestas a peticiones HTTP llevan consigo código HTML, CSS y JavaScript, podemos observador 2 particularidades importantes:

1- Con el uso de una SPA la primera petición siempre demorara más, ya que en la primera petición se transfieren todo el código HTML, CSS y de JavaScript relacionado a la página web.

2- A pesar de que la primera petición puede presentar un retardo mayor a la de un esquema clásico de cliente-servidor, todas las peticiones posteriores mostraran una mejora significativa en su velocidad de respuesta. Como se puede observar en la figura 10 estas superaran ampliamente la velocidad de aplicaciones con una arquitectura cliente-servidor clásica.

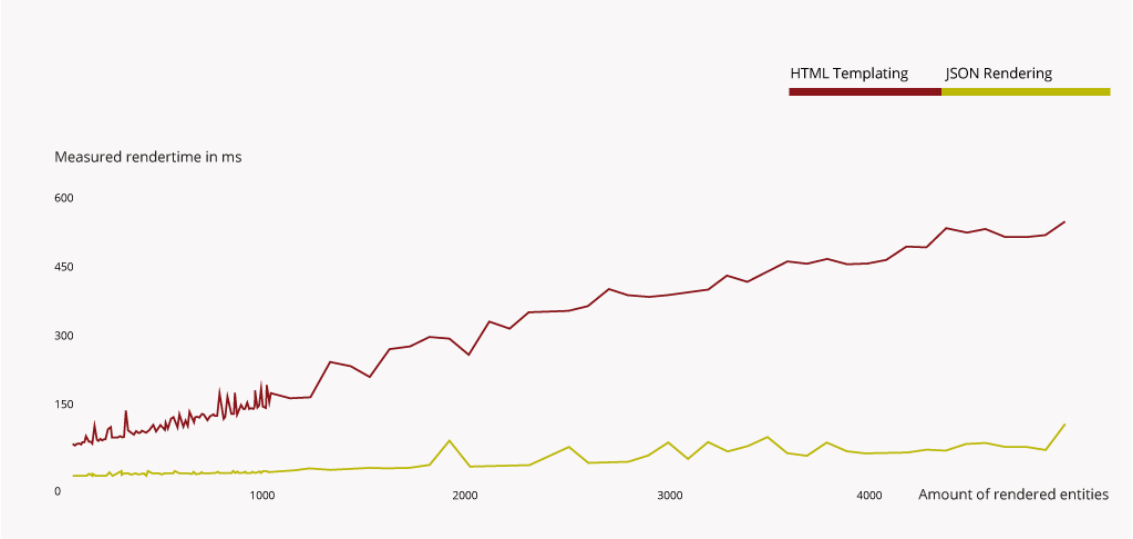

Figura 10. HTML Rendering vs JSON Rendering, [Figura]. Recuperado de: [16]

2.2 Tecnologías, Herramientas y Frameworks

A continuación se describirán características de distintas tecnologías relacionadas al desarrollo web, en particular las tecnologías mencionadas, se presentan como buenos candidatos a utilizar para la elaboración de una solución al problema planteado.

2.2.1 Lenguajes de Programación y Ambientes de Ejecución

Un lenguaje de programación es un lenguaje formal diseñado para realizar procesos que pueden ser llevados a cabo por máquinas como las computadoras. Pueden usarse para crear programas que controlen el comportamiento físico y lógico de una máquina, para expresar algoritmos con precisión, o como modo de comunicación humana.

2.2.1.1 PHP

PHP es un lenguaje de script interpretado en el lado del servidor utilizado para la generación de páginas Web dinámicas, similar al ASP de Microsoft, embebido en páginas HTML y ejecutado en el servidor. La mayor parte de su sintaxis ha sido tomada de C, Java y Perl con algunas características específicas de sí mismo. La meta del lenguaje es permitir rápidamente a los desarrollador es la generación dinámica de páginas. No es un lenguaje de marcas como podría ser HTML, XML o WML [17]. Está más cercano a JavaScript o a C, para aquellos que conocen estos lenguajes.

Características del Lenguaje

- Es fuertemente tipado (a partir de su versión 7)
- Orientado al desarrollo de aplicaciones web dinámicas
- Fácil de aprender
- Multiplataforma
- Permite aplicar técnicas de programación orientada a objetos

2.2.1.2 Hip Hop Virtual Machine (HHVM)

HipHop Virtual Machine (HHVM) es una máquina virtual de procesos basada en la compilación Just-In-Time (JIT o justo a tiempo) que sirve como un motor de ejecución para los lenguajes de programación PHP y Hack.

Al utilizar el principio de la compilación JIT, el código PHP o Hack ejecutado se transforma primero en código intermedio de bytes de Hip Hop (HHBC), que a su vez se traduce de forma dinámica a código de máquina x86-64, para luego ser optimizado y ejecutado de forma nativa. Esto contrasta con la ejecución usual del intérprete de PHP, en el cual el Zend-Engine transforma el código fuente PHP en códigos de operación que sirven de forma intermedia, y ejecuta estos directamente en el CPU virtual del Zend-Engine. [18][19]

HHVM es desarrollado por Facebook, con el código fuente del proyecto alojado en GitHub y bajo los términos de las licencias de PHP License y Zend license. [19] [20]

2.2.1.3 JavaScript

Creado por Netscape en 1995 como una extensión de HTML para Netscape Navigator 2.0, JavaScript tenía como función principal la manipulación de documentos HTML y la validación de formularios. Antes de ganar este nombre tan famoso hoy en día, JavaScript fue llamado Mocha. Cuando por primera vez enviado en las versiones beta, se llama oficialmente LiveScript y, por último, cuando fue lanzado por Sun Microsystems, fue bautizado con el nombre con el que se conoce hoy en día. Debido a los nombres similares, la gente confunde Java con JavaScript, pero no son el mismo lenguaje. A pesar de que ambos tienen la estructura léxica de la programación. Diferente de C, C # y Java, JavaScript es un lenguaje interpretado [21]. En caso de JavaScript, el intérprete solía ser solo el navegador web. Pero esto fue cambiando en los últimos años, ya que ahora JavaScript también tiene intérpretes para el lado del servidor y hasta para el desarrollo de aplicaciones de escritorio.

Características del lenguaje:

- Orientado a Objetos
- Basado en prototipos
- Imperativo
- Débilmente tipado
- Dinámico
- Asíncrono

2.2.1.4 Node.JS

Es un entorno en tiempo de ejecución multiplataforma, de código abierto, construido sobre el motor V8 de Google para la capa del servidor (pero no limitándose a ello) y está basado en la especificación de lenguaje de programación ECMAScript. Orientado a eventos asíncronos, Node.js está diseñado para construir aplicaciones de red altamente escalables [22].

Este funciona como un contenedor del motor V8 de Google, que lo optimiza para trabajar mejor en contextos distintos a un navegador web, y además provee una serie de API's optimizados para ciertos casos de uso. A su vez, Node.js funciona en un ambiente totalmente asíncrono basado en eventos y no bloqueante, lo cual permite realizar muchas operaciones de entrada/salida con un costo mínimo de tiempo, comportamiento usual y deseado en aplicaciones web.

Node.js se registra con el sistema operativo y cada vez que un cliente establece una conexión se ejecuta un callback. Dentro del ambiente de ejecución de Node.js, cada conexión recibe una pequeña asignación de espacio de memoria dinámico, sin tener que generar un hilo de trabajo [22]. A diferencia de otros servidores dirigidos por eventos, el lazo de manejo de eventos de Node.js no es llamado explícitamente sino que se activa al final de cada ejecución de una función de callback. El lazo de manejo de eventos se termina cuando ya no quedan eventos por atender.

También incorpora varios "módulos básicos" compilados en el propio binario, como por ejemplo el módulo de red, que proporciona una capa para programación de red asíncrona y otros módulos fundamentales, como por ejemplo Path, FileSystem, Buffer, Timers y el de propósito más general Stream. Es posible utilizar módulos desarrollados por terceros, ya sea como archivos ".node" pre compilados, o como archivos en Javascript plano. Los módulos Javascript se implementan siguiendo la especificación CommonJS para módulos, utilizando una variable de exportación para dar a estos scripts acceso a funciones y variables implementadas por los módulos.

2.2.2 Node Package Manager (NPM)

NPM es básicamente una manera sencilla de compartir código JavaScript que fue creado para resolver problemas particulares, para que así sea fácil para otros desarrolladores reutilizar ese código en sus propias aplicaciones [23]. Además vuelve sencillo el manejo de versiones de estas distintas soluciones, ya que te permite actualizar de manera sencilla cada solución. Estas soluciones suelen ser llamadas paquetes o módulos. Los paquetes son simplemente directorios con uno o más archivos dentro de él, además de un archivo "package.json", este archivo especifica los paquetes que esta solución en particular utiliza.

Una aplicación completa puede estar compuesta de cientos de paquetes, ya que la idea principal de estos paquetes es que sean pequeños y ofrezcan una solución puntual a un problema específico.

Las funcionalidades de NPM le permiten a un equipo de desarrollo importar módulos específicos de otros equipos que sean concentrados en la solución de problemas recurrentes. Además de que permite reciclar código de proyectos o soluciones anteriores que mostraron buenos resultados.

2.2.3 Express.JS

Express es un framework minimalista de aplicaciones web flexibles para Node.js, que proporciona un robusto conjunto de características para la web y aplicaciones móviles [24]. Con una gran variedad de métodos de utilidad HTTP y middleware a su disposición, crear una potente API resulta rápido y fácil. Express proporciona una capa delgada de características fundamentales de aplicaciones web, sin afectar las características y rendimiento de Node.js.

Es importante señalar que "minimalista" se refiere a que Express.js tiene como objetivo mantener el núcleo simple pero extensible. Este no toma muchas decisiones tecnológicas por el desarrollador, tales como la base de datos, ORM u ODM a utilizar. Y las decisiones que si hace por el desarrollador, son fáciles de editar. Todo lo demás depende de que

necesite el desarrollador, de modo que Express.js puede ser todo lo que necesites y nada que no desees. Esto junto con NPM vuelve a las aplicaciones y proyectos desarrollados a través de Express altamente modulares.

2.2.4 Angular.JS

HTML es ideal para declarar documentos estáticos, pero se tambalea cuando tratamos de utilizarlo para declarar vistas dinámicas en aplicaciones web. AngularJS le permite ampliar el vocabulario HTML para su aplicación. El medio ambiente resultante es expresivo, legible y rápido de desarrollar [25].

AngularJS es un framework de desarrollo de aplicaciones web que usa tecnologías del lado del cliente (HTML, CSS, Javascript). Es desarrollado y mantenido por Google. Es importante destacar de acuerdo a Brad Green y Shyam Seshadri (2013) que AngularJS se basa en el esquema de una aplicación de una sola página (SPA), donde básicamente nunca se hace una carga completa de la página sino de secciones específicas usando técnicas como AJAX. Para el desarrollo se basa en el patrón MV*, muy parecido a Modelo Vista Controlador, solo que en este caso el "Controlador" puede ser cualquier otro tipo de componente.

Otros frameworks se ocupan de las deficiencias de HTML ya sea abstrayéndose de HTML, CSS y / o JavaScript o proporcionando una manera imperativa para manipular el DOM. Ninguno de estos aborda el problema de raíz el cual es que HTML no fue diseñado para las vistas dinámicas. AngularJS al permitir ampliar HTML ataca este problema.

2.2.5 Electron

Electron es un framework formado por un conjunto de tecnologías web que te permite crear aplicaciones de escritorio con JavaScript puro, proporcionando un ambiente de ejecución con un diverso API nativo. Se podría ver como una variante del motor de ejecución Node.js que se centra en las aplicaciones de escritorio en lugar de los servidores web.

Electron funciona a través de dos procesos:

- Proceso Principal: Es el proceso que ejecuta los distintos módulos relacionados a la aplicación, desde este proceso se pueden mostrar páginas web.
- Proceso Renderizador: Desde que Electron utiliza Cromium para la visualización de páginas web, también se utiliza la arquitectura multi-proceso de Cromium. Cada página web de Electron se ejecuta en su propio proceso, a este se le llama proceso renderizador.

En los navegadores normales, las páginas web generalmente se ejecuta en un entorno de espacio aislado y no se les permite el acceso a los recursos nativos. En Electron, sin embargo, tienen el poder de utilizar las API Node.js en las páginas web que permiten las interacciones con el sistema operativo a nivel inferior [26].

2.2.6 Bootstrap

Bootstrap es un framework de Javascript de código abierto desarrollado por el equipo de Twitter. Es una combinación de HTML, CSS y Javascript código diseñado para ayudar a construir los componentes de interfaz de usuario responsivos [27]. Bootstrap también se programó para apoyar tanto HTML5 y CSS3.

También denominado como un framework de frontend. Bootstrap es un conjunto de herramientas gratuitas para la creación de aplicaciones web. Contiene HTML y plantillas de diseño basadas en CSS para tipografía, formas, botones, navegación y otros componentes de la interfaz, así como extensiones de JavaScript opcionales.

Algunas de sus características más resaltantes son:

- Muy fácil de aprender
- Enfoque de desarrollo basado en celdas (grid system)
- Estilos predeterminados para muchos componentes básicos de HTML
- Plugins de JavaScript

2.2.7 Git

Es un programa para el manejo de repositorios de código y sus versiones. Este provee un número importante de funcionalidades para la manipulación de estos repositorios y versiones. Este básicamente toma una fotografía del estado actual de todos sus archivos en ese momento y almacena una referencia a esa "foto". Para ser eficaz, si los archivos no han cambiado, Git no almacena el archivo de nuevo, sólo un enlace al archivo idéntico anterior que ya ha almacenado [28].

Git es capaz de manejar las distintas situaciones que pueden presentarse para mantener data congruente, resuelve conflictos, realiza mezclas (merge) de distintas versiones de código si le es posible, he indica que existen conflictos entre las versiones si no le es posible. Se podría pensar en Git como un pequeño sistema de archivos muy poderoso, que permite guardar un historial de las distintas versiones de los archivos que ha manipulado.

2.2.8 MongoDB

MongoDB es uno de varios tipos de bases de datos que surgió a mediados de la década de 2000 bajo la etiqueta NoSQL. En lugar de utilizar tablas y filas como en las bases de datos relacionales, MongoDB está construido sobre una arquitectura de colecciones y documentos. Los documentos comprenden conjuntos de pares de valores clave y son la unidad básica de datos en MongoDB [29]. Colecciones contienen conjuntos de documentos y funcionan como el equivalente de tablas de bases de datos relacionales.

Al igual que otras bases de datos NoSQL, MongoDB soporta el diseño del esquema dinámico, permitiendo que los documentos en una colección tener diferentes campos y estructuras. La base de datos utiliza un formato de almacenamiento de documentos e intercambio de datos llamada BSON, que proporciona una representación binaria de documentos JSON. Su fragmentación automática de las colecciones permite que estas estén distribuidas a lo largo de múltiples sistemas, esto para asegurar la escalabilidad horizontal mientras que el volumen de datos crezca.

2.2.9 Mongoose.JS

Mongoose.JS es un módulo de Node.JS que implementa un ODM (Object-Document Mapping) para la integración de este con el sistema manejador de base de datos MongoDB. Este provee varias funcionalidades para facilitar la construcción de consultas a la base de datos y explotar al máximo posible las ventajas de que ofrece MongoDB.

2.2.10 NodeGit

NodeGit provee un API con todas las funcionalidades más básicas de Git para facilitar la integración de la Herramienta con Node.JS, este evita brindar las funcionalidades de más alto nivel desarrolladas en Git para dar la posibilidad a los desarrolladores de implementarlas de la mejor manera posible dentro del contexto de interés para ellos.

2.2.11 Request

Este módulo de JavaScript para proyectos de construidos sobre Node.JS hace posible la utilización de llamadas HTTP a otros servicios web, mediante el uso de un API sencillo que permite el uso de los distintos métodos HTTP y el manejo de sus respuestas.

2.2.12 Aglio

Aglio es un módulo que crea vistas HTML a partir de archivos que cumplan con un formato BluePrint, este suele ser utilizado para generar documentación de software. Aglio ha demostrado ser particularmente conveniente para generar documentación de API REST, ya que brinda una sintaxis específica para la elaboración de estos.

2.2.13 UI Ace

Dentro de los requerimientos de la aplicación de manejo de contenido (CMA) se encuentra la posibilidad de editar el código asociado a los proyectos, para esto es necesaria la utilización de un editor de código que provea todas las funcionalidades básicas necesarias para esto.

Ace es un editor de Código integrable desarrollado en JavaScript, que proporciona todas las funcionalidades básicas de un editor de código y UI Ace es un módulo de JavaScript para Angular.JS que brinda una forma sencilla de integrar las funcionalidades del Ace con un Proyecto desarrollado en Angular.JS.

2.2.14 UI Bootstrap

UI Bootstrap es un módulo que hace posible la integración de las funcionalidades de JavaScript que ofrece Bootstrap con el framework Angular.JS, esto mediante la declaración de distintas directivas (elementos HTML extendidos a través de Angular.JS) que otorgan dichas funcionalidades.

2.2.15 textAngular

Dentro de toda página o aplicación web existe una determinada cantidad texto necesario para brindar información de algún tipo, textAngular es un módulo para el framework Angular.js que facilita la edición de texto al proveer un conjunto funcionalidades básicas para la edición de este.

2.2.16 Pilas de Desarrollo

2.2.16.1 XAMP

XAMP ha sido una de las pilas de desarrollo más usadas en el mundo web a lo largo de los años. Su nombre es acrónimo de los nombres de sus componentes que son:

- Linux, Windows o MacOS (Entre otros): El sistema operativo.
- Apache: El servidor web.
- MySQL/MariaDB: Sistema manejador de base de datos relacional.
- PHP: Lenguaje de programación.

Pueden existir ligeras variantes de esta pila, al utilizar otros lenguajes o sistemas manejadores de bases de datos como Python o MongoDB.

2.2.16.2 MEAN

Es otra de las pilas de desarrollo web con mucho interés actualmente, es más novedosa que LAMP debido al poco tiempo que tienen sus componentes en el mercado. Una de las características que más resalta de MEAN es que todos los componentes comparten el lenguaje de programación Javascript y el lenguaje de comunicación JSON (Javascript Notation Object). Lo que hace de esta pila de desarrollo un ambiente muy flexible para el trabajo grupal en la tarea de desarrollo de software.

Sus componentes son los siguientes:

- MongoDB: Una base de datos no relacional enfocada en rapidez y alta escalabilidad.
- Express.js: Framework minimalista para desarrollo aplicaciones web (backend).
- Angular.js: Framework MV* para el desarrollo de SPAs (frontend).
- Node.js: Plataforma para realizar aplicaciones en lado del servidor.

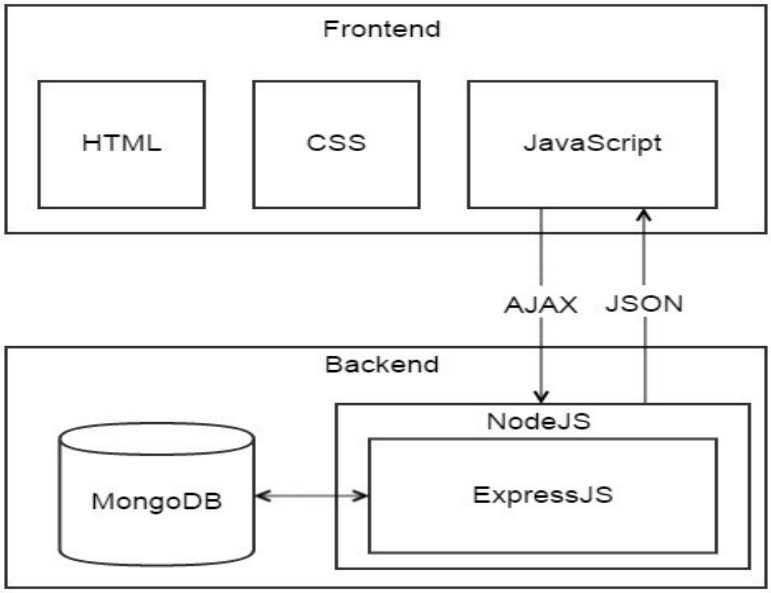

Figura 11. Pila de desarrollo MEAN, "Elaboración Propia"

CAPITULO III: Marco Metodológico

3.1 Metodología SCRUM

Un proyecto Scrum implica un esfuerzo de colaboración para crear un nuevo producto, servicio, o cualquier otro resultado como se define en la Declaración de Visión de Proyecto (Project Vision Statement) que se define en un principio [30]. Los proyectos se ven afectados por las limitaciones de tiempo, costo, alcance, calidad, recursos, capacidades organizativas, y otras limitaciones que los hacen difíciles de planificar, ejecutar, administrar y finalmente tener éxito. Sin embargo, la implementación exitosa de los resultados de un proyecto acabado le proporciona ventajas económicas significativas a una organización. Por lo tanto, es importante que las organizaciones seleccionen y practiquen una metodología adecuada de gestión de proyectos. Scrum es una de las metodologías ágiles más populares. Es una metodología de adaptación, iterativa, rápida, flexible y eficaz, diseñada para ofrecer un valor significativo de forma rápida en todo el proyecto. Scrum garantiza transparencia en la comunicación y crea un ambiente de responsabilidad colectiva y de progreso continuo. El marco de Scrum, está estructurado de tal manera que es compatible con los productos y el desarrollo de servicio en todo tipo de industrias y en cualquier tipo de proyecto, independientemente de su complejidad.

Una fortaleza clave de Scrum radica en el uso de equipos multi-funcionales, auto organizados, y con poder que dividen su trabajo en ciclos de trabajo cortos y concentrados llamados Sprints (ver Figura 12).

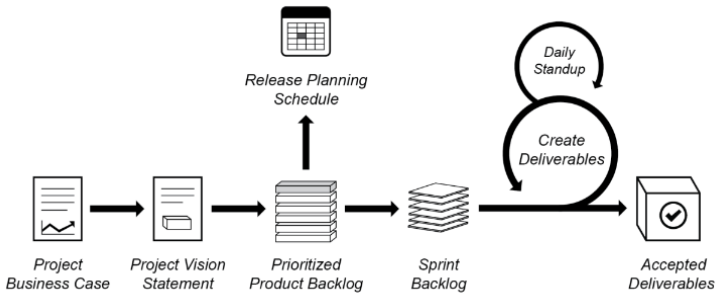

Figura 12. SCRUMstudy. (2016). Flujo de SCRUM para un Sprint [Figura]. Recuperado de: [30]

El ciclo de Scrum comienza con una reunión de Stakeholders (Stakeholder Meeting), durante el cual se crea la visión del proyecto. El Dueño del Producto, entonces desarrolla un Prioritized Product Backlog que contiene una lista priorizada de los requerimientos del negocio en forma de historia de usuario (User Story). Cada Sprint comienza con una reunión para su planeación durante la cual los requerimientos o historias de usuario (User Stories) de alta prioridad son considerados para su inclusión en el Sprint. Un Sprint suele durar entre una y seis semanas en el cual el Scrum Team trabaja en la creación de Entregables (Deliverables) potencialmente listos en incrementos del producto. Durante el Sprint, se llevan a cabo reuniones diarias (Daily Standup Meetings) cortas y muy concretas donde los miembros del equipo discuten progresos diarios. A medida que concluye el Sprint, una reunión de planeación de sprint (Sprint Planning Meeting) se lleva a cabo en el cual al Dueño del Producto y a los Stakeholders relevantes se les proporciona una demostración de los bienes y servicios desarrollados. El Dueño del Producto acepta las entregas sólo si cumplen con los criterios de aceptación (Acceptance Criteria) predefinidos. El ciclo de Sprint termina con un Retrospect Sprint Meeting, donde el equipo presenta modos para mejorar los procesos y el rendimiento a medida que avanzan al siguiente Sprint.

Algunas de las ventajas principales de la utilización de Scrum en cualquier proyecto son [30]:

- Adaptabilidad: Control de Proceso Empírico y Entregas iterativas, hacen que los proyectos sean adaptables y abiertos a la incorporación del cambio
- Transparencia: Todos los radiadores o fuentes de información tal como un Scrumboard y Sprint Burndown Chart son compartidos, lo que lleva a un ambiente de trabajo abierto.
- Retroalimentación continua: Retroalimentación continua se proporciona a través de los procesos llamados Conduct Daily Standup y Demonstrate and Validate Sprint.
- Mejoramiento Continuo: Los entregables se mejoran progresivamente Sprint por Sprint a través del proceso Groom Prioritized Product Backlog.
- Entrega Continúa de Valor: Los procesos iterativos permiten la entrega continua de valor tan frecuentemente como el Cliente lo requiere a través del proceso Ship Deliverable.

- Paso sostenible: Los procesos Scrum están diseñados de tal manera que las personas involucradas pueden trabajar a un paso cómodo o sostenible que, en teoría, se puede continuar indefinidamente.
- Entrega Anticipada de Alto Valor: El proceso de Create Prioritized Product Backlog asegura que los requisitos de mayor valor del Cliente sean los primeros en cubrirse.
- Proceso de Desarrollo Eficiente: holguras en los tiempos y la reducción al mínimo de trabajo que no es esencial conduce a mayores niveles de eficiencia.
- Motivación: Los procesos de Conduct Daily Standup y Restrospect Sprint conducen a mayores niveles de motivación
- Resolución de problemas de Forma más Rápida: Colaboración y colocación de equipos multi-funcionales conducen a la resolución de problemas con mayor rapidez.
- Entregables Efectivos: El proceso de Create Priotized Product Backlog y revisiones periódicas después de la creación de entregables asegura entregas efectivas para el cliente.
- Centrado en el Cliente: El poner énfasis en el valor del negocio y tener un enfoque de colaboración con los stakeholders asegura un marco orientado al cliente.
- Entorno de alta confianza: Los procesos de conduct daily standup and retrospect sprint promueven transparencia y colaboración, dando lugar a un ambiente de trabajo de alta confianza asegurando así una baja fricción entre los empleados.
- Responsabilidad Colectiva: El proceso de Approve, Estimate and Commit User Stories permite que los miembros del equipo se sientan responsables del proyecto y su trabajo resultando en una mejor calidad.
- Alta velocidad: Un marco de colaboración que le permite a los equipos multifuncionales altamente cualificados alcanzar su potencial y alta velocidad.
- Medio Ambiente innovador: Los procesos Retrospect Sprint y Retrospect Project crean un ambiente de introspección, aprendizaje y capacidad de adaptación que lleva a un entorno de trabajo innovador y creativo.

En la parte anterior se mencionaron varias etapas del ciclo de la metodología SCRUM. Se puede decir que las más importantes son:

- Planificación del Backlog: consiste en la definición de un documento con los requisitos del sistema identificados por prioridad. En esta fase se realiza la planificación del "Sprint 0" en donde se definen los objetivos y el trabajo a realizar en dicha iteración. De esta reunión se obtiene un Sprint Backlog, el cual contiene una lista de tareas y el objetivo más relevante del Sprint.

- Seguimiento del Sprint: consta de reuniones diarias en donde surgen tres preguntas principales para la evaluación de los avances de las tareas, dichas preguntas son:
    a. ¿Qué trabajo fue realizado desde la reunión anterior?
    b. ¿Qué trabajo se realizaran hasta la siguiente reunión?
    c. ¿Qué inconvenientes han surgido, como se solucionan para continuar?

- Revisión del Sprint: es la última fase donde se realiza una evaluación del sprint, considerando el incremento alcanzado. Los resultados finales alcanzados son presentados y se genera una versión de muestra con el fin de constatar con el cliente el correcto cumplimiento de las funcionalidades, en caso de existir discrepancias, se aplican los correctivos pertinentes.

3.1.1 Roles

El equipo Scrum abarca tres roles: el dueño del producto, el equipo de desarrollo y el maestro Scrum. Los equipos cumplen con ser auto-organizados y multifuncionales, es decir, eligen la mejor forma de llevar a cabo un trabajo y tienen todas las competencias necesarias para cumplir con el trabajo sin depender de agentes externos al equipo.

Se describen a continuación los distintos roles que se desarrollan en los equipos Scrum:

- Dueño del producto: es el responsable de maximizar el valor del producto y del trabajo del equipo de desarrollo y es la única persona responsable de gestionar la pila de producto. Esta tarea varía ampliamente de acuerdo a la organización, el equipo Scrum y el individuo.

- Equipo de desarrollo: son los profesionales encargados de entregar un incremento del producto potencialmente usable al final de cada Sprint, siendo solos los miembros del equipo de desarrollo quienes participan en la creación del incremento. Los equipos de desarrollo se estructuran y reciben poderes por parte de la organización para gestionar su propio trabajo. La sinergia resultante optimiza la eficiencia y efectividad general del equipo de desarrollo.

- Maestro Scrum: es el responsable de asegurar el entendimiento y cumplimiento de la metodología, asegurándose de que el equipo trabaja ajustándose a la teoría, prácticas y reglas de SCRUM. El maestro Scrum es un líder servil, al servicio del equipo.

3.1.2 Artefactos

Los artefactos definidos por Scrum están específicamente diseñados para maximizar la transparencia de la información clave, que es necesaria para asegurar que el equipo tenga éxito al entregar un incremento, proporcionando oportunidades para la inspección y adaptación del proyecto.

Los tres artefactos que provee la metodología SCRUM son los siguientes:

- Pila de producto: es una lista ordenada de todo lo que podría ser necesario en el producto, y es la única fuente de requerimientos para cualquier cambio a realizarse sobre dicho producto. El dueño de producto es el responsable de la pila de producto, incluyendo su contenido, disponibilidad y ordenación.

- Pila de Sprint: es el conjunto de elementos de la pila de producto seleccionados para el Sprint, más un plan para entregar el incremento de producto y conseguir el objetivo del Sprint. La pila de Sprint es una predicción hecha por el equipo de desarrollo acerca de qué funcionalidad formará parte del próximo incremento y del trabajo necesario para entregar esa funcionalidad [30].

- Incremento: se trata de la suma de todos los elementos de la pila de producto completados durante un Sprint y durante todos los Sprint previos. Al final de un Sprint, la nueva parte del incremento debe estar completada, lo cual significa que está en condiciones de ser utilizada y que cumple la definición del equipo Scrum.

CAPITULO IV: Marco Aplicativo

4.1 Implementación de la Metodología SCRUM

Se tomaran en cuenta varios de los aspectos considerados por SCRUM, pero también se simplifican otros aspectos dado al tamaño reducido del equipo de desarrollo y el alcance del proyecto. Los principales cambios en la Metodología empleada se describen a continuación.

4.1.1 Roles

Los roles del Dueño del Producto y el Maestro SCRUM para el desarrollo del presente TEG, son ejercidos por el tutor académico y el Ingeniero Charles Ochoa respectivamente. El Equipo de Desarrollo está compuesto por dos miembros, el estudiante desarrollando el TEG, el cual estará encargado de todo el desarrollo e implementación de la solución y el Diseñador Jesús Poleo, cuya única responsabilidad es supervisar el debido seguimiento de las guías de diseño de interfaces para el desarrollo de la herramienta. No se consideran terceras partes involucradas.

4.1.2 Artefactos

El principal artefacto que se tomará de SCRUM es la Pila del Producto ya que la misma plantea los requerimientos funcionales a desarrollar necesarios para completar los objetivos específicos planteados en las siguientes secciones (4.2). También se fijará una meta Sprint por cada iteración la cual consistirá en una descripción breve de la meta a lograr en cada iteración.

Numero de Sprint	Tarea(s)	Duración (Semanas)
0	Levantamiento de Requerimientos y Diseño de la Solución	2
1	Desarrollo y Documentación de la Aplicación de Entrega de Contenido	4
2	Desarrollo de la Aplicación de Manejo de Contenido	6

Tabla 1: Pila del Producto [Tabla]. "Elaboración Propia"

4.1.3 Reuniones

De las reuniones planteadas por SCRUM sólo se llevaran a cabo la Reunión de Planificación Sprint y la Reunión de Revisión Sprint, incluyendo la evaluación retrospectiva en esta misma. Debido a que el equipo de trabajo está compuesto solamente por dos integrantes se descartan las reuniones diarias, ya que uno de estos integrantes solo supervisara el desarrollo de las interfaces de usuario. Además, la Reunión de Planificación Sprint de la siguiente iteración, se realizara inmediatamente después de concluida la Reunión de Revisión Sprint de la iteración anterior.

4.2 Sprint 0: Levantamiento de Requerimientos y Diseño de la Solución

Meta del Sprint: Captar todos los requerimientos necesarios para el debido inicio del desarrollo del proyecto.

4.2.1 Requerimientos

Los distintos requerimientos captados para el desarrollo de un CMS se pueden dividir en los dos módulos que lo componen, tomando esto en cuenta dichos requerimientos serian:

- La aplicación de Entrega de Contenido (CDA):
    - Debe cumplir con la convención REST.
    - Manejo de Usuarios mediante el uso de las operaciones CRUD básicas.
    - Autenticación de dichos usuarios y Manejo de distintos niveles de privilegio según el tipo de este.
    - Manejo de Proyectos a través de las operaciones CRUD básicas y operaciones específicas para manejar la instanciación de estos en los distintos ambientes de desarrollo y producción.
    - Manejo de las distintas versiones del proyecto, manteniendo la coherencia al integrar distintos cambios proveniente de múltiples fuentes.
    - Brindar accesibilidad a las distintas versiones del proyecto a cualquier instanciación de la Aplicación de Manejo de Contenido.

- Manejo de Operaciones CRUD para la manipulación de los repositorios locales asociados a los proyectos.

- La aplicación de Manejo de Contenido (CMA):

  - Consumo de todos los Servicios ofrecidos por la CDA.
  - Manejo de versiones de los proyectos localmente.
  - Generación de contenido en tiempo real.
  - Interfaces para todas las interacciones que implican un consumo de servicios de la CDA.
  - Interfaz para creación y modificación de contenido que muestre el funcionamiento de este en tiempo real y que permita la edición código del proyecto.

Tomando en cuenta los distintos requerimientos planteados se elaboró un diagrama de casos de uso para la mejor visualización de estos (Se puede consultar las especificaciones de dichos casos de uso en la sección de Anexos).

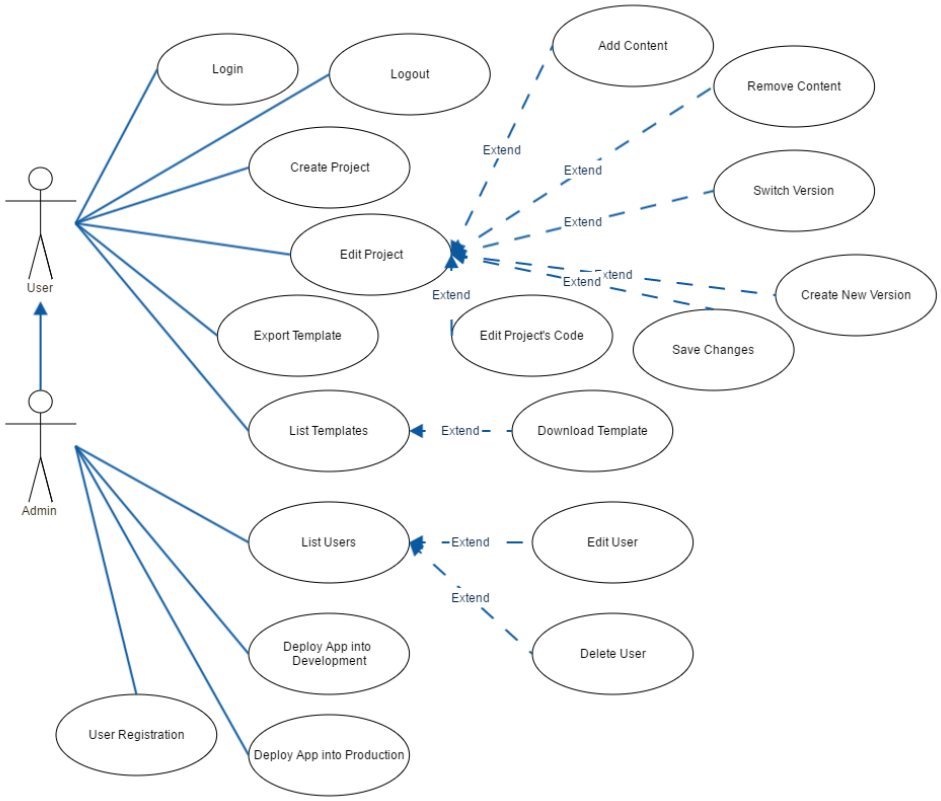

Figura 13. Requerimientos para Platypus. (2016). Diagrama de Casos de Uso [Figura]. "Elaboración Propia"

Habiendo establecido los distintos requerimientos para la captación de funcionalidades del aplicativo se planteó el uso de las siguientes tecnologías anteriormente descritas para el desarrollo de este.

Para el desarrollo de la Aplicación de Entrega de Contenido:

- Node.JS: Principalmente por su excelente manejo de concurrencia y velocidad para servir datos.
- Express.JS: Ya que nos provee un conjunto de funcionalidades y tecnologías para construcción rápida y sencilla de un Web API.
- MongoDB: Por su esquema sumamente flexible y tolerancia a los cambios.

Para el desarrollo de la Aplicación de Manejo de Contenido:

- Angular.JS: Ya que brinda un conjunto convenciones para la separación de la lógica del front-end del back-end, además de contar con una gran cantidad de módulos desarrollados para la reutilización de soluciones probadas.
- Electron: Principalmente porque brinda funcionalidades para acceder al sistema de archivos del sistema operativo host, cualidad necesaria dentro de la aplicación de manejo de contenido para hacer posible el manejo de archivos y sus versiones a través de Git. Razón principal por la cual se escogió que la CMA fuera una aplicación de escritorio.

Para la integración y el funcionamiento debido de ambas:

- Git: Ya que nos brinda un conjunto bastante completo de funcionalidades para el manejo de versiones de archivos y código.
- Bitbucket: Gracias a que nos brinda la capacidad de manejar las versiones de distintos archivos a través de Git y de manera remota, para brindar accesibilidad ilimitada a los distintos proyectos desarrollados con la herramienta.

4.2.2 La Solución

En base a la pila de desarrollo generada y los distintos requerimientos mencionados en la sección anterior, se propone que el aplicativo a desarrollar posea la arquitectura que se puede observar en la figura 13.

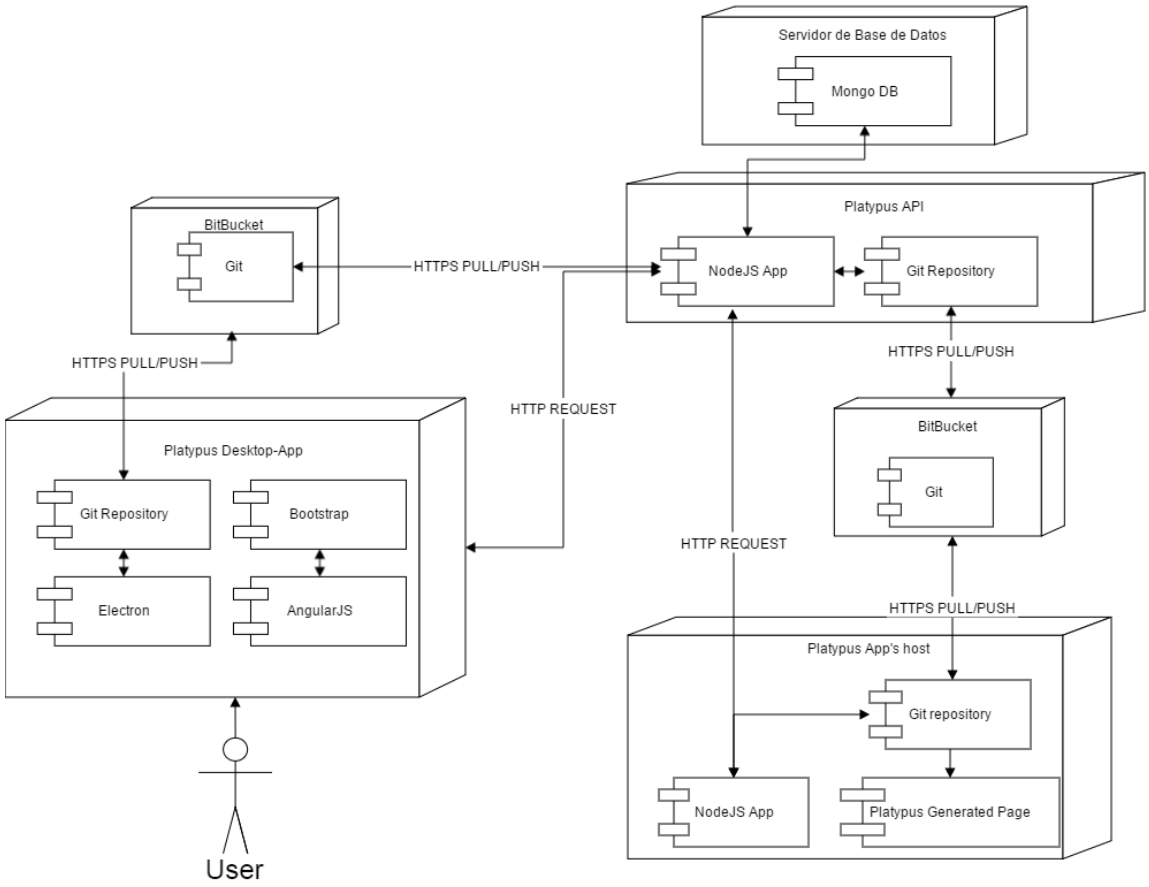

Figura 14. Arquitectura de Platypus. (2016). Diagrama de Despliegue [Figura]. "Elaboración Propia"

La arquitectura mostrada en la figura 13 muestra los distintos módulos del aplicativo y como estos se comunican. Los componentes mostrados en la parte izquierda de la imagen corresponden a la aplicación de manejo de contenido (CMA), mientras que los componentes vistos en la derecha corresponden a la aplicación de entrega de contenido (CDA). Básicamente lo que esta arquitectura busca es aprovechar al máximo la utilización de repositorios de código a través de Git y Bitbucket, no solo brindando la capacidad de

manejar distintas versiones de los proyectos generados por la herramientas sino también de brindar una capa extra de seguridad entre el ambiente de desarrollo (CMA) y el ambiente de producción (CDA), utilizando un repositorio centralizado distinto (una instancia de Bitbucket distinta) para cada uno, manteniendo así las versiones de proyecto listas para el ambiente de producción separadas de las que siguen en desarrollo.

Para explicar de manera más específica el funcionamiento de la CDA planteada, en la figura 14 podemos observar que esta está compuesta por dos módulos distintos:

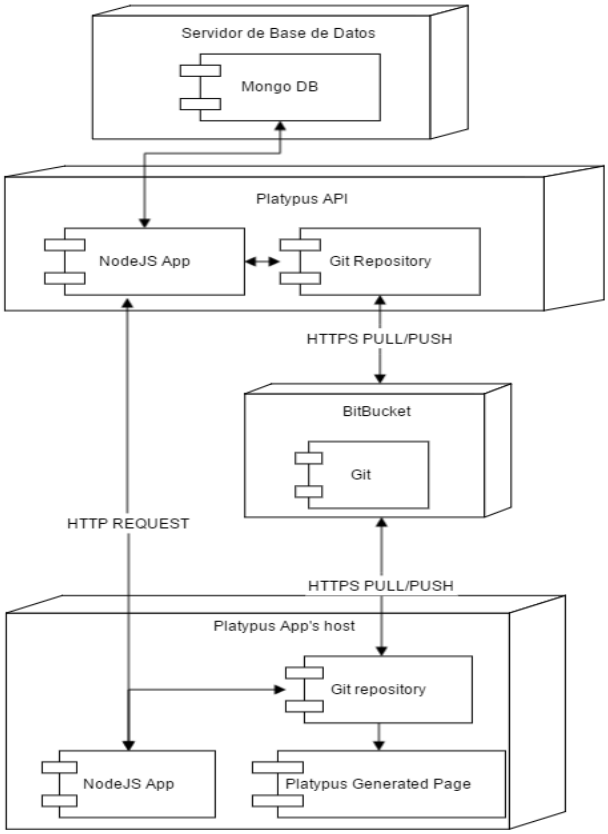

Figura 15. Arquitectura de Platypus. (2016). Diagrama de Despliegue: Módulo CDA [Figura]. "Elaboración Propia"

- Platypus API: Este módulo estará encargado de la comunicación con la CMA y con los repositorios centralizados (Ambas instancias de Bitbucket), además del manejo de usuarios y la creación de proyectos de la herramienta. También será una interfaz entre la CMA y el módulo Platypus App's Host.

- Platypus App's Host: Este módulo se comunica con un repositorio de código que solo manejara instancias de proyectos listos para entrar en el ambiente de producción y sus funcionalidades solo pueden ser accedidas a través del módulo Platypus API. Además maneja toda la configuración de archivos necesaria para que los proyectos sean visibles en la Web.

Luego al observar la figura 15 podemos apreciar que la CMA cuenta con un único modulo, este está compuesto por Electron, Angular.js y repositorios de Git. A través de Angular.js se generaran todas las vistas y se manejara la experiencia de usuario, Electron por su lado, facilitara el manejo de archivos/repositorios tanto locales como de Bitbucket y el acceso a las funcionalidades del sistema operativo que sean necesarias.

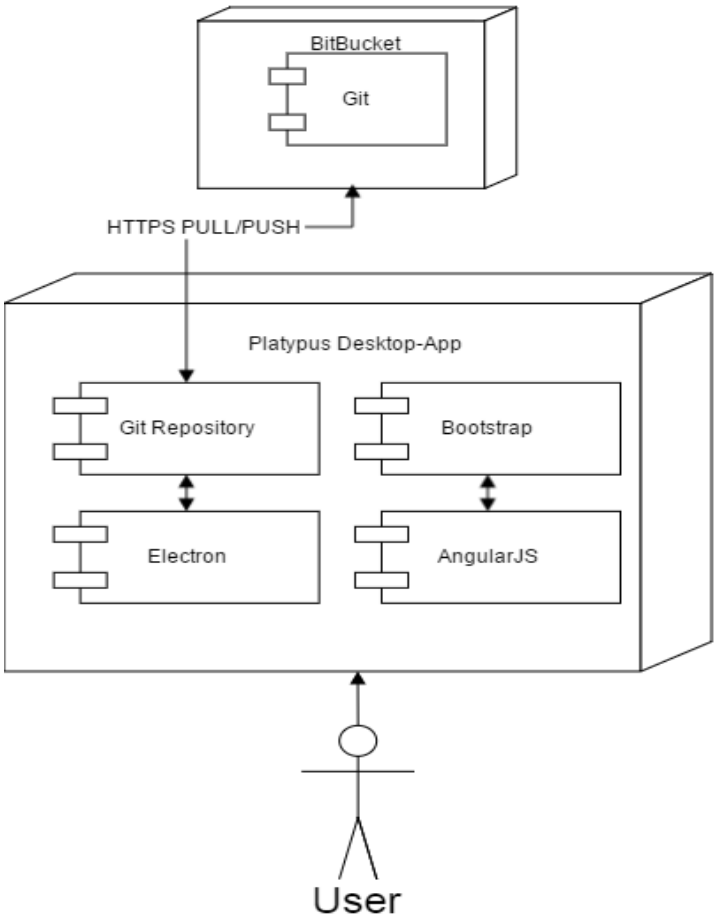

Figura 16. Arquitectura de Platypus. (2016). Diagrama de Despliegue: Módulo CMA [Figura]. "Elaboración Propia"

4.3 Sprint 1: Desarrollo y Documentación de la Aplicación de Entrega de Contenido (CDA)

Meta del Sprint: Desarrollar los Módulos/Web APIs que conforman la CDA.

La primera tarea dentro del objetivo de desarrollo de los módulos que componen la CDA fue la elaboración de un diagrama de objetos del dominio (ver figura 17), para tener una mejor visualización de les entidades involucradas.

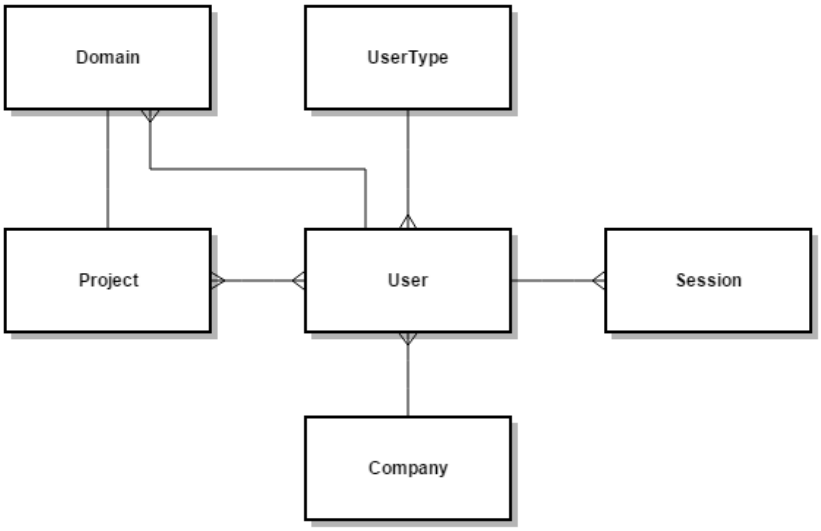

Figura 17. Documentación Platypus (2016). Diagrama de Objetos del Dominio [Figura]. "Elaboración Propia"

A continuación se dará una breve explicación de las entidades planteadas en la figura 17 junto con papel que cumplen dentro de la solución planteada:

- User/Usuario: Entidad que representa a los distintos usuarios de la aplicación, los cuales tienen la capacidad de hacer uso de distintas funcionalidades como:
  - Consulta, creación, edición y eliminación de proyectos y otros usuarios.
  - Capacidad para cambiar el estado de los usuarios para negar o brindar acceso a la aplicación.
  - Consulta, creación y eliminación de dominios.
  - Consulta de tipos de Usuario.
  - Asociación de Proyectos a distintos usuarios y viceversa.

Todo esto por medio del uso de un token de Sesión El modelo de usuario posee los siguientes atributos: nombre, apellido, nombre de usuario, email, tipo, contraseña, lista de proyectos asociados, lista de dominios asociados, compañía a la que pertenece y su estado en el sistema.

- Tipo de Usuario/UserType: Esta entidad es para el manejo interno y solo esta compuesta por un "nombre".
- Proyecto/Project: Entidad que representa la instancias de los proyectos manejados por los usuarios de la herramienta. Dentro de la edición de la información de estos proyectos existen las siguientes opciones:
    - Cambiar el proyecto de ambiente, ya sea de local a desarrollo o de desarrollo a producción.
    - Cambiar su estado a público para hacerlo visible a todos los usuarios de la herramienta.

  El modelo de los proyectos está compuesto por los siguientes atributos: nombre, su dirección física en el servidor, ambiente en el que encuentra, lista de usuarios asociados, dominio, su estado en el sistema y si este es público o no.
- Sesión/Session: Entidad que representa la instancia de una sesión de usuario en la aplicación, esta puede ser creada o eliminada y posee los atributos: Usuario al que pertenece la sesión y token.
- Compañía/Company: Entidad que representa la compañía que utiliza la aplicación y que engloba a todos los usuarios relacionados a esta en un ambiente cerrado para brindar una mayor capa de seguridad a los proyectos de dicha compañía. Posee los atributos: nombre, dirección y descripción.
- Dominio/Domain: Entidad que representa los Dominios asociados a las aplicaciones web desarrolladas de para la compañía. Posee el atributo "nombre".

Luego de definir las distintas entidades necesarias para el debido funcionamiento de la CDA se pasó a la definición de las distintas Rutas o Servicios que esta prestaría, manteniendo la convención REST para el diseño de estas. En la figura 18 podemos observar algunas estas.

```
65  //User Routes
66  router.route('/user')
67    .get(Session.validate, User.isAdmin, User.listAll)
68    .post(Session.validate, User.isAdmin, User.create);
69
70  router.route('/user/me')
71    .get(Session.validate, User.listOne)
72    .patch(Session.validate, User.softDelete)
73    .put(Session.validate, User.update);
74
75  router.route('/user/:idUser')
76    .patch(Session.validate, User.isAdmin, User.softDelete)
77    .put(Session.validate, User.isAdmin, User.update);
```

Figura 18. Documentación Platypus (2016). Algunas Rutas de Usuario [Figura]. "Elaboración Propia"

4.3.1 Módulos

Para el desarrollo de la CDA, se utilizaron varios módulos que resolvían desde necesidades muy puntuales a necesidades muy generales, a continuación se listan los módulos que fueron de mayor importancia:

- Mongoose.JS: Para la integración de MongoDB con el Framework Express.JS y Node.JS.
- NodeGit: Para la integración de la herramienta Git con con Express.JS y Node.JS.
- Request: Para hacer posible el uso de las llamadas HTTP que comunican los dos módulos que componen la CDA entre sí y con el API de Bitbucket.
- Aglio: Para la generación de la documentación del Web API.

4.3.2 Implementación del Modelo de Datos

Una vez fueron integrados todos los módulos necesarios para el desarrollo de la Aplicación de Entrega de Contenido (CDA), se procedió a implementar el modelo de datos que se planteó a través del diagrama de Objetos del dominio de la figura 17.

4.3.2.1 Roles

Dentro del modelo de Datos se definieron dos posibles tipos de usuario, los cuales son:

- Usuario: Este puede crear, modificar y publicar proyectos, además puede enviar los proyectos al ambiente de desarrollo.
- Administrador: Este puede crear, modificar y eliminar tanto proyectos como usuarios, también puede asociar proyectos a otros usuarios o enviarlos a los ambientes de desarrollo y producción.

4.3.3 Implementación y documentación de los Servicios

Una vez implementado el modelo, se procedió a desarrollar los distintos servicios y funcionalidades de la CDA. A continuación se nombraran algunos de los servicios implementados junto con una breve descripción de la tarea que realizan:

Servicio	Métodos	Funcionalidad
/user	GET y POST	GET: Lista todos los usuarios POST: Crea un Usuario
/user/:idUser/project	POST	Asocia a un proyecto al usuario cuyo Id fue especificado
/project/:idProject/development	PUT	Realiza las operaciones de Git sobre el repositorio del proyecto correspondiente para ponerlo en el ambiente de desarrollo
/project/:idProject/production	PUT	Realiza las operaciones de Git sobre el repositorio del proyecto correspondiente para ponerlo en el ambiente de Producción, además de crear los archivos de configuración necesarios para el proyecto sea visible en la Web
/project/:idProject	PATCH y DELETE	PATCH: Activa o desactiva el proyecto de los distintos ambientes en los que se encuentre. DELETE: Elimina el Proyecto de todos los ambientes en los que se encuentre

/me/project/branch	GET	Permite ver todas las versiones creadas de un proyecto

Tabla 2: Lista de Algunos de los Servicios de Platypus API [Tabla]. "Elaboración Propia"

Una vez desarrollados los servicios, se pasó a elaborar toda la documentación referente a estos. Primero elaborando un archivo en formato BluePrint que cumpliera la sintaxis requerida por aglio, para luego ejecutar la herramienta y generar el documento a HTML que especifica el funcionamiento de las distintas rutas de los módulos de la CDA (ver imagen 21).

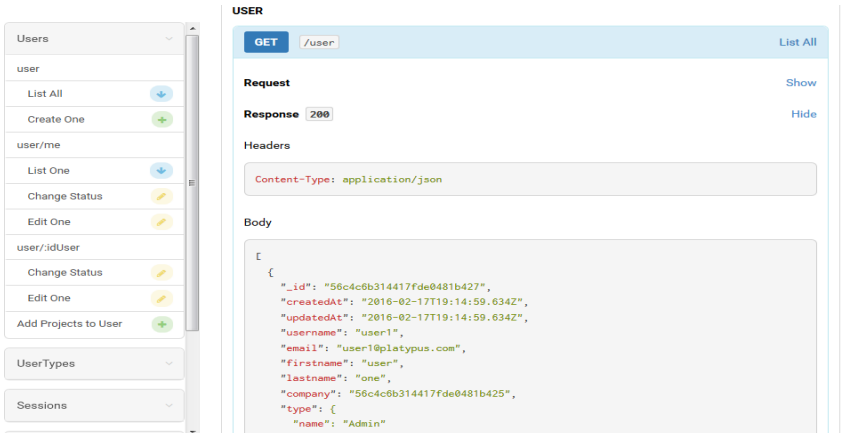

Figura 19. Documentacion de Platypus (2016). Documentacion de Servicios [Figura]. Recuperado de: http://platydev.inworknet.com:3005/api#users

4.3.4 Conclusiones del Sprint 1

Al final del Sprint 1 se cumplieron todos los objetivos de manera satisfactoria y se plantearon nuevas funcionalidades que pueden agregar valor al producto en futuras expansiones. Algunas de estas fueron:

- Capacidad para manipular el servidor de DNS asociado a la Aplicación de Entrega de Contenido, para así poder agregar nuevos dominios o sub-dominios.
- Brindar funcionalidades para la exportación de proyectos que se encuentran en repositorios ajenos a la herramienta.

4.4 Sprint 2: Desarrollo de la Aplicación de Manejo de Contenido (CMA)

Como se mencionó anteriormente, las tecnologías escogidas para el desarrollo de la CMA, fueron:

- Angular.JS
- Electron
- Bootstrap
- Git

El primer paso en el desarrollo de la CMA fue realizar la integración de estas tecnologías, para poder lograr esto fue necesaria la utilización de ciertos módulos de JavaScript.

4.4.1 Módulos

Dentro de los módulos necesarios para la integración de las tecnologías mencionadas anteriormente los más importantes fueron:

- UI Ace: Para la integración del editor de código Ace al aplicativo siendo desarrollado.
- UI Boostrap: Para hacer posible el uso de las funcionalidades de JavaScript que ofrece bootstrap.
- TextAngular: Para mejorar la experiencia de usuario y brindar un editor de texto intuitivo en la interfaz de la CMA.

4.4.2 Integración de los Servicios de la Aplicación de Entrega de Contenido

Una vez todas las tecnologías anteriormente mencionadas fueron integradas de manera exitosa, el siguiente paso del desarrollo consistió en la integración de los distintos servicios brindados por la aplicación de entrega de contenido. Para esto se desarrollaron distintos servicios de Angular.JS.

Cabe a destacar que en un servicio de Angular.JS es la forma en la que se representan los modelos y la lógica de negocio en los desarrollos apoyados por este framework. Los servicios básicamente consisten en la implementación de un API que contenga todas las llamadas HTTP necesarias para comunicarse con los distintos servicios que sirvan datos o funcionalidades a la aplicación, además de otra funcionalidades creadas para el formateo o procesamiento de datos necesario para mostrar estos de la manera deseada en las vistas de la aplicación.

Para la implementación de Aplicación de Manejo de Contenido fue necesaria la implementación de cinco servicios distintos, los cuales son:

- User Service: Servicio responsable de todas las llamadas HTTP y funcionalidades necesarias para el manejo de usuarios de la herramienta.
- Project Service: Servicio responsable del Manejo de Proyectos, este implementa no solo llamadas HTTP sino también manejo de archivos relevantes el proyecto.
- Git Service: En este servicio se utilizó NodeGit para la construcción de las distinta funcionalidades necesarias para el manejo de repositorios de código, algunas de estas fueron: “add”, “commit”, “push”, “pull”, “merge”, “fetch”, “checkout” entre otras.
- Modal Service: En este servicio se definieron las distintas vistas modales necesarias para la correcta interacción con el usuario, tales como:
    - Mensajes de Error
    - Mensajes de Espera
    - Formularios
    - Mensajes de Transición
- Templates Service: En este servicio se definieron distintas plantillas HTML para el manejo controlado de la generación de contenido. Este servicio lleva el control de los distintos componentes generados por el usuario en un proyecto brindando funcionalidades para la creación y modificación de dichos componentes.

4.4.3 Implementación de Directivas

Una directiva es la forma en que el framework de Angular.JS extiende elementos o atributos HTML mediante JavaScript, estas son recomendadas para desarrollar funcionalidades que modifiquen al DOM y que tengan cierto nivel de complejidad. Las directivas otorgan mayor capacidad de extensibilidad al código ya que no solo impulsan la reutilización de este sino que también lo hacen más expresivo.

Dentro de los requerimientos del desarrollo de la Aplicación de Manejo de contenido, existe la necesidad de agregar contenido dinámicamente al proyecto en el que se esté trabajando, en pro de brindar una mejor experiencia de usuario se propuso el desarrollo de una directiva para que brinde una funcionalidad de arrastrado (Drag-and-Drop) a los distintos elementos de contenido que conforman el proyecto.

Mediante el uso del Servicio *Template Service* mencionado anteriormente la directiva desarrollada es capaz de insertar el HTML correspondiente al contenido arrastrado sobre la interfaz del proyecto (ver figura 23).

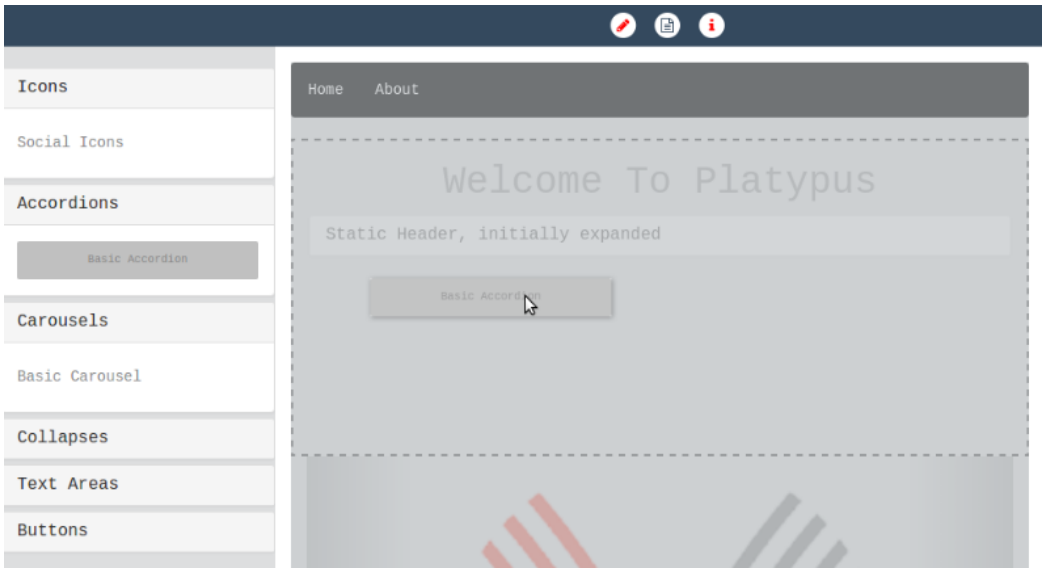

Figura 20. Vista de La Aplicación (2016). Drag-and-Drop [Figura]. "Elaboración Propia"

Como se puede observar en la figura 24 una vez se realiza el arrastre del elemento, se levanta una ventana modal que brinda distintas opciones para debido llenado del contenido que fue desplazado sobre el proyecto.

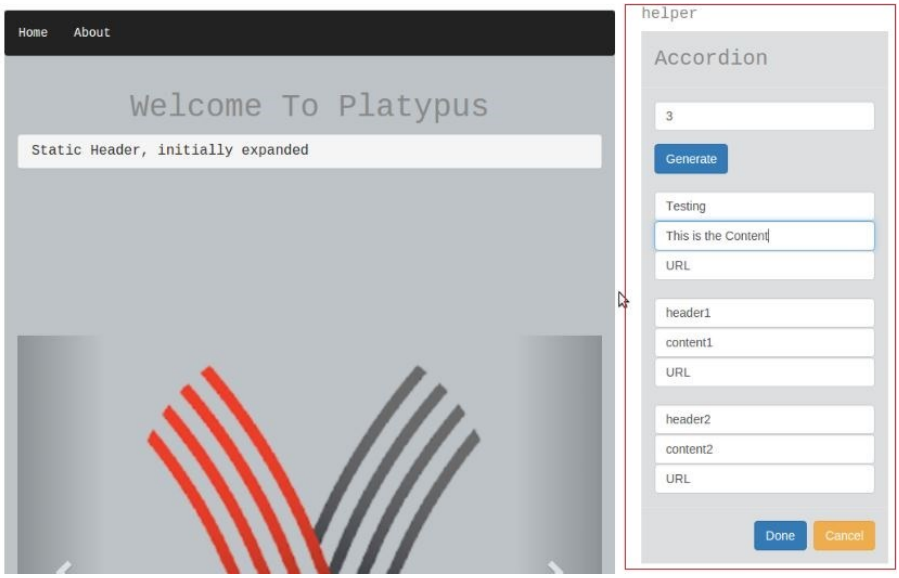

Figura 21. Vista de La Aplicación (2016). Edición del Contenido [Figura]. "Elaboración Propia"

Luego al observar la figura 25 podemos ver el efecto que tuvo el llenado del formulario de la vista de ayuda para edición del contenido.

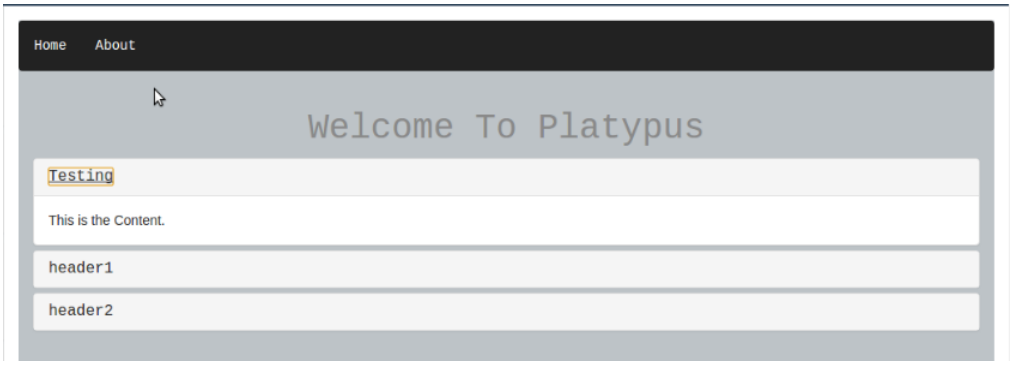

Figura 22. Vista de La Aplicación (2016). Contenido Modificado [Figura]. "Elaboración Propia"

4.4.4 Desarrollo de las Vistas de la Aplicación de Manejo de Contenido

A continuación se dará un breve resumen de algunas de las vistas desarrolladas para la aplicación de manejo de contenido, haciendo énfasis solo en las funcionalidades más relevantes de la aplicación

4.4.4.1 Login

Esta vista provee un formulario básico para la autenticación del usuario dentro del CMS, mas no provee ninguna funcionalidad para el registro de dichos usuarios, ya que esta funcionalidad esta delegada únicamente a los administradores de la herramienta (ver figura 26).

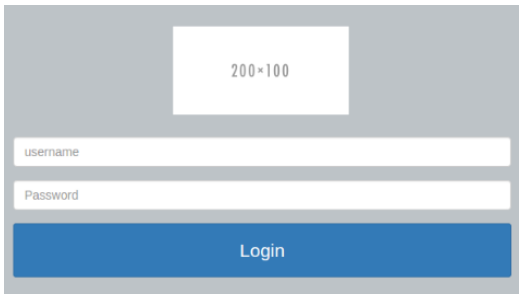

Figura 23. Vista de La Aplicación (2016). Login [Figura]. "Elaboración Propia"

4.4.4.2 Dashboard

Esta vista permite la visualización de todos los Proyectos al que el usuario tiene acceso, además de la opción de crear nuevos y es un punto de partida para la navegación hacia todas las funcionalidades de la herramienta (ver figura 27).

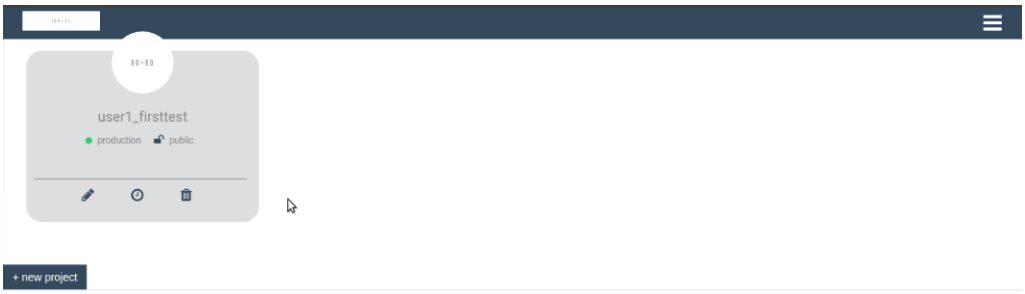

Figura 24. Vista de La Aplicación (2016). Dashboard [Figura]. "Elaboración Propia"

En la figura 28 al ver con más detalle la zona de la interfaz que denota el detalle del proyecto. Podemos ver las distintas opciones que esta ofrece, las cuales son:

- Estado que refleja el Ambiente en el que encuentra el Proyecto
- Estado público o privado que denota la capacidad de otros usuarios de la herramienta de descargar el proyecto como template o tema.
- Icono clickleable para la edición del proyecto
- Icono clickleable para la eliminación del proyecto
- Icono señalable que indica su última fecha de edición

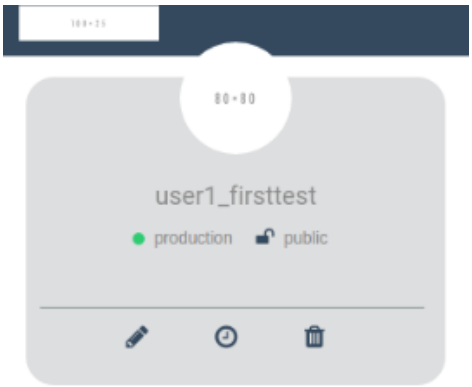

Figura 25. Vista de La Aplicación (2016). Dashboard-Detalle de Proyecto [Figura]. "Elaboración Propia"

Por ultimo en la figura 29 podemos las opciones ofrecidas por el Navbar del Dashboard. Opciones que facilitan la navegación a otras funcionalidades de la aplicación.

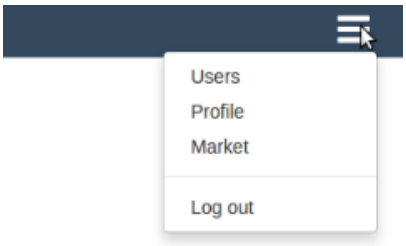

Figura 26. Vista de La Aplicación (2016). Dashboard-Opciones del navbar [Figura]. "Elaboración Propia"

4.4.4.3 Market

El Market es una vista en la que se listan todos los proyectos que pueden ser descargados para usar como punto de partida para la elaboración de nuevos proyectos (ver figura 30), también llamados temas en otros CMS, estos proyectos promueven la reutilización de soluciones generadas por la herramienta.

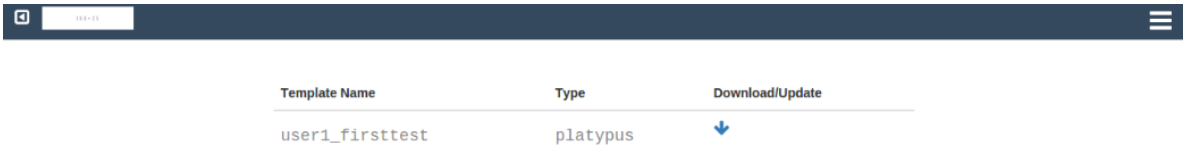

Figura 27. Vista de La Aplicación (2016). Market [Figura]. "Elaboración Propia"

4.4.4.4 Administración de Usuarios

Existen cuatro vistas dentro de la aplicación para el manejo de usuarios:

- Perfil de Usuario
- Listado de Usuarios
- Creación de Usuario
- Edición de Usuarios

A continuación se explicara brevemente la vista de edición de usuario, ya que se considera la más relevante dentro del uso de la funcionalidades brindadas por la aplicación de entrega de contenido.

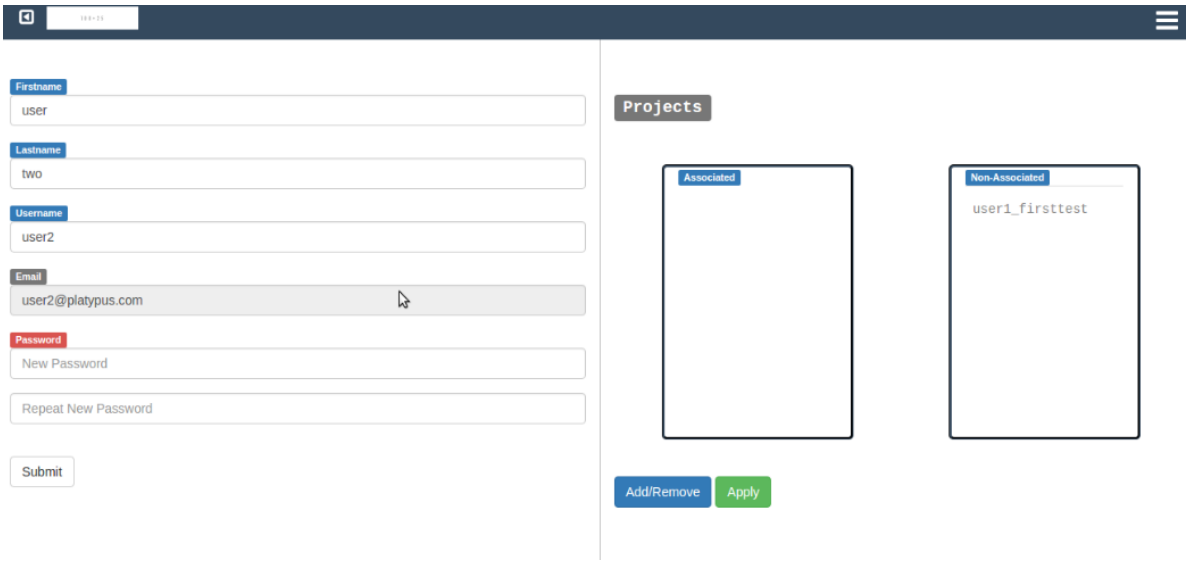

Figura 28. Vista de La Aplicación (2016). Edit User [Figura]. "Elaboración Propia"

En la figura 31 se pueden apreciar dos secciones, en la izquierda un formulario que permite editar la información del usuario seleccionado y en la derecha una interfaz que permite la visualización de los proyectos asociados y no asociados al usuario, además de brindar la posibilidad de asociar o desasociar dichos proyectos al usuario en cuestión.

4.4.4.5 Administración de Proyectos

Además de las funcionalidades para la visualización de estados y creación de proyectos ya vista en el Dashboard, existe otra vista que otorga distintas opciones para editar la información o estados referentes a un proyecto.

Figura 29. Vista de La Aplicación (2016). Edit Project-1 [Figura]. "Elaboración Propia"

En la figura 32 se puede observar que en la parte izquierda de la vista es posible visualizar la información del proyecto junto con el estado de este en los distintos ambientes de desarrollo y producción, mientras que en lo zona derecha se brinda una interfaz para asociar y desasociar usuarios a dicho proyecto.

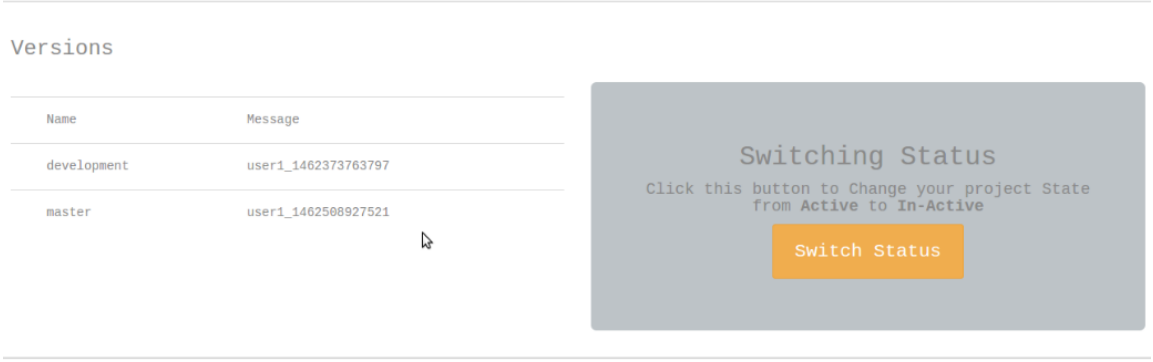

Figura 30. Vista de La Aplicación (2016). Edit Project-2 [Figura]. "Elaboración Propia"

Mientras que en la figura 33 podemos observar las distintas versiones que existen del proyecto, además de tener la opción de activar y desactivar la aplicación web asociada a este sí y solo si el proyecto se encuentra en un ambiente de producción.

4.4.4.6 Editor

Dentro de las distintas vistas que componen la aplicación, el editor resulta la más resaltante, ya que contiene el mayor número funcionalidades importantes para la creación y modificación de contenido en los proyectos. A continuación se explicaran las partes más importantes de esta vista.

Figura 31. Vista de La Aplicación (2016). Barra de Herramientas del Editor [Figura]. "Elaboración Propia"

En la figura 34 podemos observar la barra de herramientas perteneciente al editor, esta provee un conjunto de funcionalidades vitales para la debida manipulación del proyecto. Descritas de arriba hacia abajo y de izquierda a derecha estas funcionalidades son:

- Projects: Brinda las opciones para guardar los distintos cambios realizados en el proyecto en cualquiera de los ambientes, además de ofrecer funcionalidades para la creación de contenido (ver figura 35).

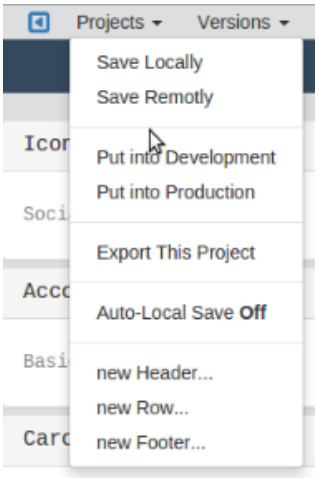

Figura 32. Vista de La Aplicación (2016). Barra de Herramientas del Editor, Primera Opcion [Figura]. "Elaboración Propia"

- Versions: Da la capacidad de crear nuevas versiones del proyecto además de navegar entre dichas versiones.

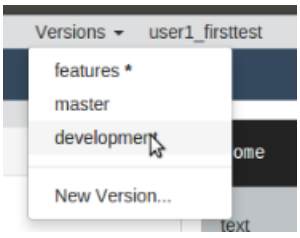

Figura 33. Vista de La Aplicación (2016). Barra de Herramientas del Editor, Segunda Opcion [Figura]. "Elaboración Propia"

- La siguientes 3 funcionalidades están representadas por iconos, y nos permiten ocultar y mostrar 3 elementos distintos de esta vista:

    - La vista Previa de la Aplicación que está siendo desarrollada (ver figura 37).

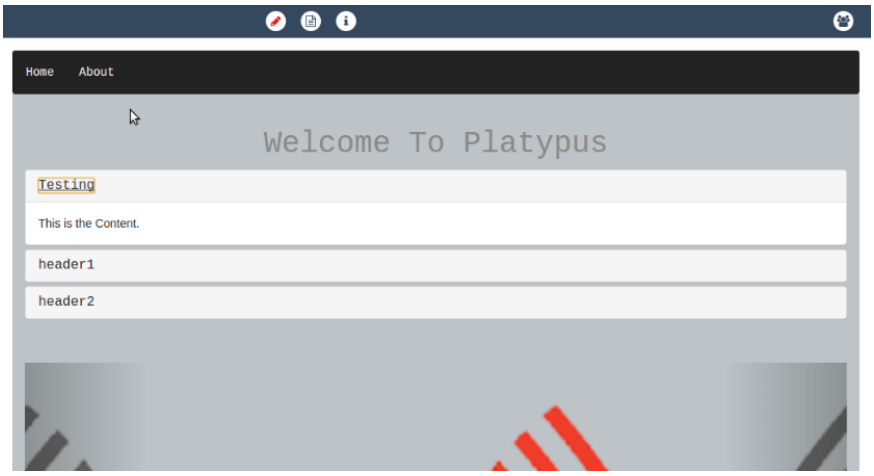

Figura 34. Vista de La Aplicación (2016). Vista previa [Figura]. "Elaboración Propia"

    - El editor de código para la edición a más bajo nivel de los elementos de la página web en desarrollo (ver figura 38).

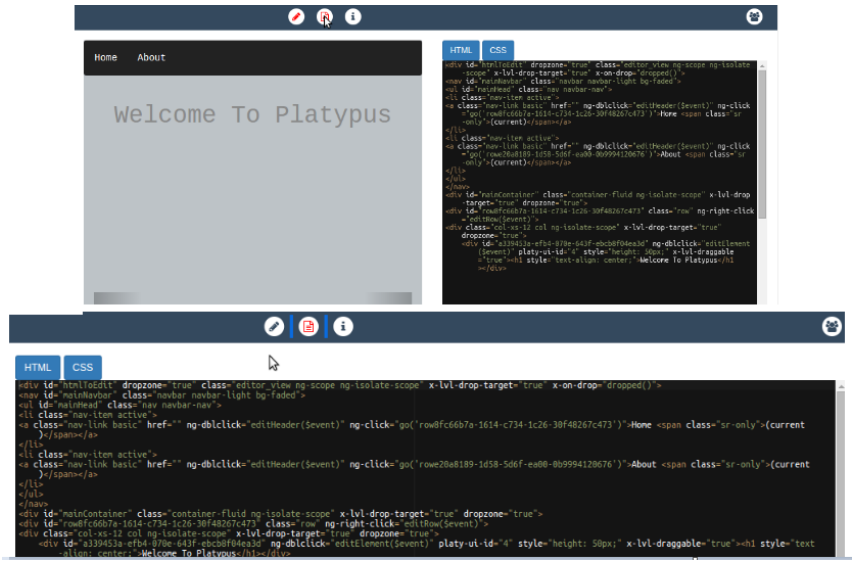

Figura 35. Vista de La Aplicación (2016). Editor de Código [Figura]. "Elaboración Propia"

- Y una vista auxiliar que muestra distintos formularios para la edición del contenido que se arrastra a la vista previa (ver figura 39).

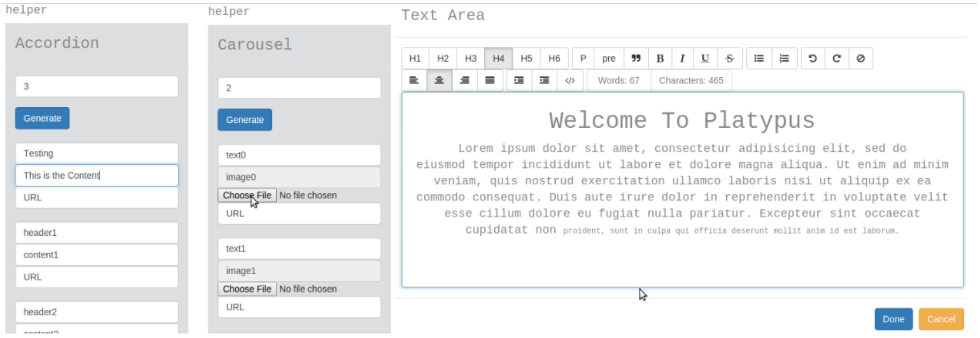

Figura 36. Vista de La Aplicación (2016). Vistas de Ayuda para la edición de contenido [Figura]. "Elaboración Propia"

Por último, otra parte importante de la vista de editor se puede observar en la figura 40 la cual muestra un listado de componentes o contenidos que pueden ser arrastrados (Drag-and-Drop) sobre la vista previa del proyecto para agregarlos de manera dinámica a este.

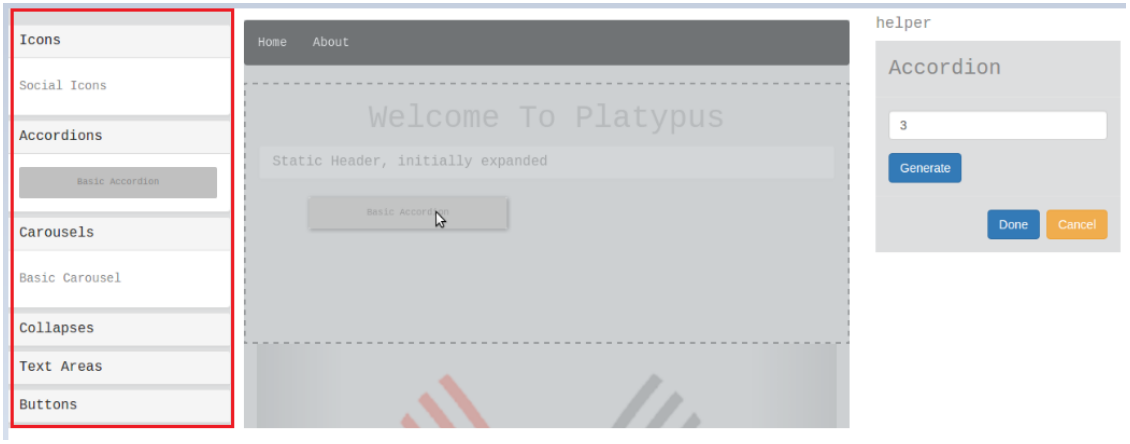

Figura 37. Vista de La Aplicación (2016). Listado de Contenido [Figura]. "Elaboración Propia"

4.4.4.7 Conclusiones del Sprint 2

Como conclusión del Sprint 2 se puede decir que todos los objetivos fueron alcanzados satisfactoriamente, ya que todas las funcionalidades planteadas fueron desarrolladas exitosamente, particularmente los servicios *Git Service* y *Template Service* son las funcionalidades que aportan más valor al aplicativo, ya que una provee de todas las funciones necesarias para el manejo de repositorios de código y la otra provee un API fácilmente extensible para la generación y edición de contenido.

Como recomendación para futuras expansiones del proyecto, se plantea la extensión del servicio *Template Service* para brindar más opciones de contenido y funcionalidades a los usuarios de la herramienta.

4.5 Resultados

Como resultado de este desarrollo se obtuvo un producto de software con conjunto de funcionalidades básicas con la posibilidad de ser extendidas de manera sencilla y que facilitaran el futuro crecimiento de la herramienta.

Para dar una breve demostración de las funcionalidades de la herramienta, a continuación se elaborara una página web sumamente sencilla a través de la aplicación y se explicara el uso de las funcionalidades en cada paso.

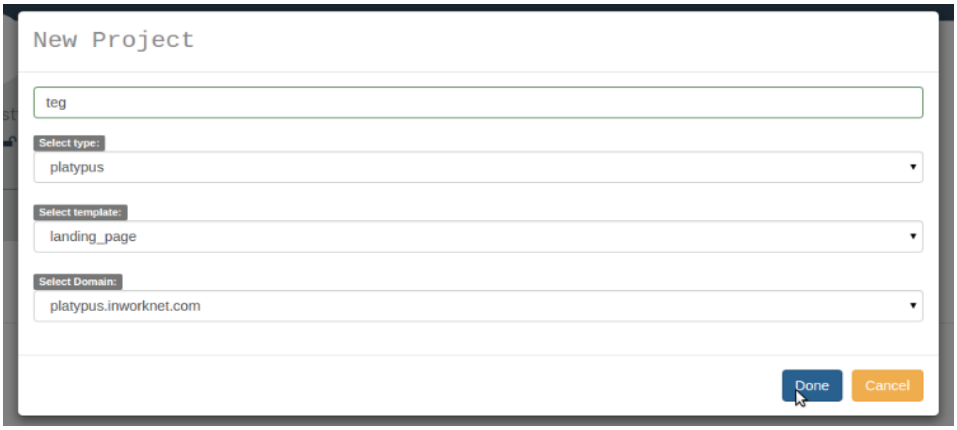

Figura 38: Creacion de Proyecto [Figura]. “Elaboración Propia”

Para empezar en la figura 41 podemos ver como se crea una instancia de proyecto a través de un formulario en el que se especifica el nombre, el tipo, el template a utilizarse y el dominio que el proyecto ocupara una vez este esté es producción.

Figura 39: Definición de Navbar [Figura]. "Elaboración Propia"

Luego en la figura 42 podemos ver cómo una vez creado el proyecto surgirán ventanas modales para especificar elementos básicos (En este caso secciones del Navbar) de la página web a ser desarrollada, en las figuras 43 y 44 se puede ver como se definen el tamaño y distribución de las columnas de la primera fila y los iconos y enlaces del footer o sección inferior respectivamente.

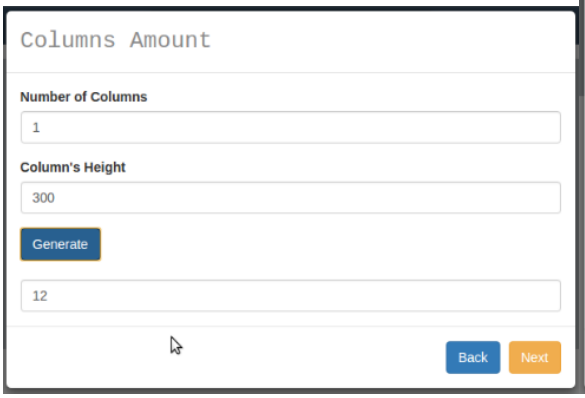

Figura 40: Definición de Primera Columna

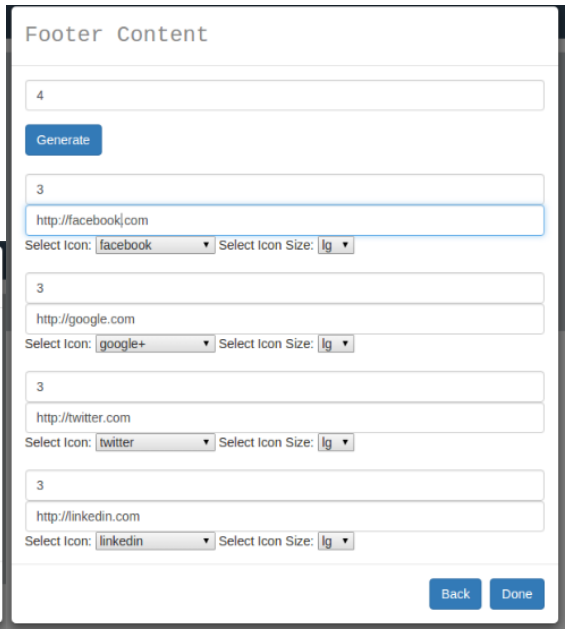

Figura 41: Definición del Footer

Una vez se definieron los elementos básicos en los pasos anteriores, podemos apreciar a través la figura 44 una primera vista del proyecto en desarrollo

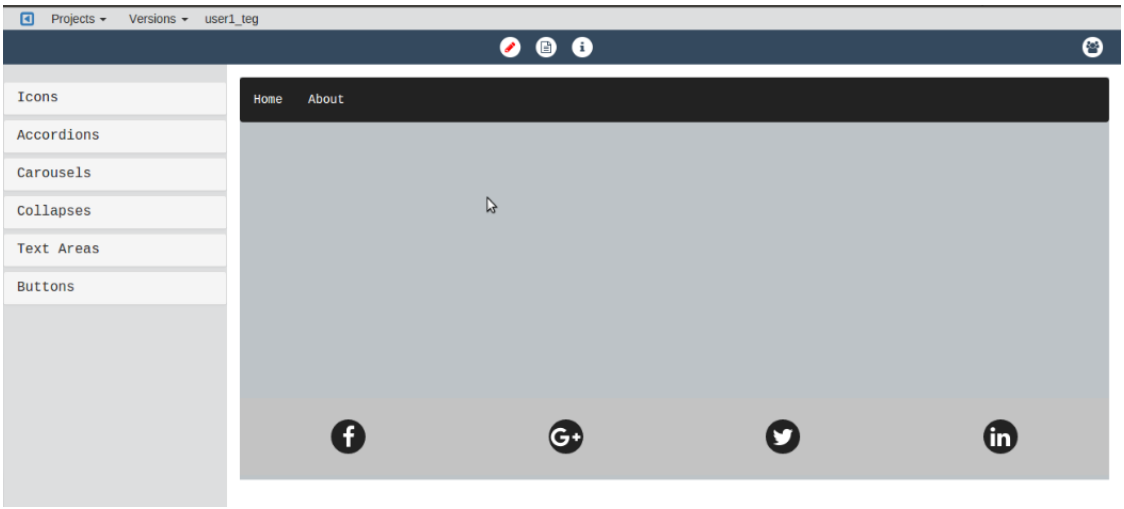

Figura 42: Vista previa del proyecto [Figura]. "Elaboración Propia"

Ahora procederemos a agregar distintos contenido a la página web:

- Figura 46 se agrega un Mensaje de texto en la primera fila de la página web
- Figura 47 se agrega una nueva fila dividida en 3 columnas, con una proporción 3-6-3 (siendo 12 el máximo de columnas por fila) para resaltar el contenido del medio
- Figura 48 se agrega un carousel con 3 imágenes en la columna del centro, en la parte derecha de la imagen se puede apreciar el formulario para la configuración del carousel

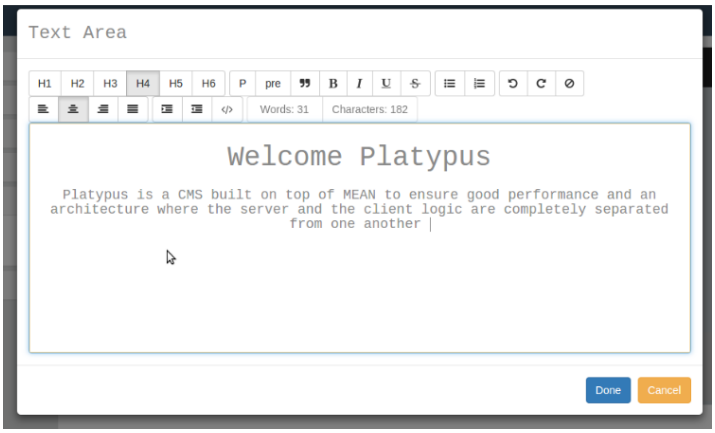

Figura 43: Modal para la edición de Texto [Figura]. "Elaboración Propia"

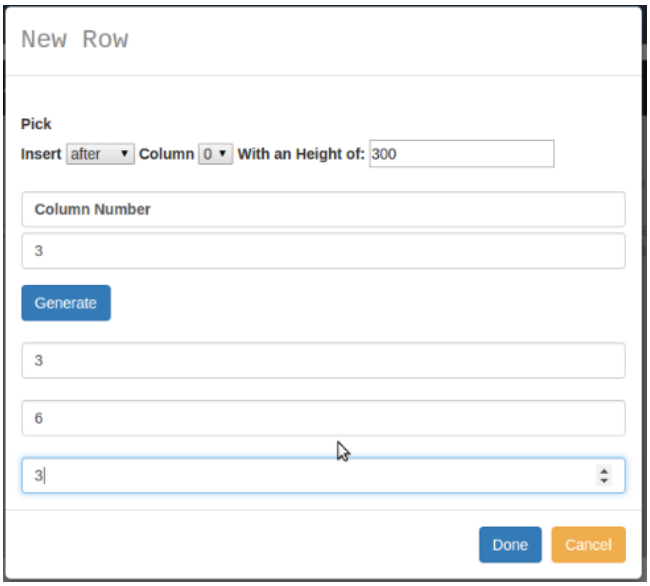

Figura 44: Agregando una Nueva Fila [Figura]. "Elaboración Propia"

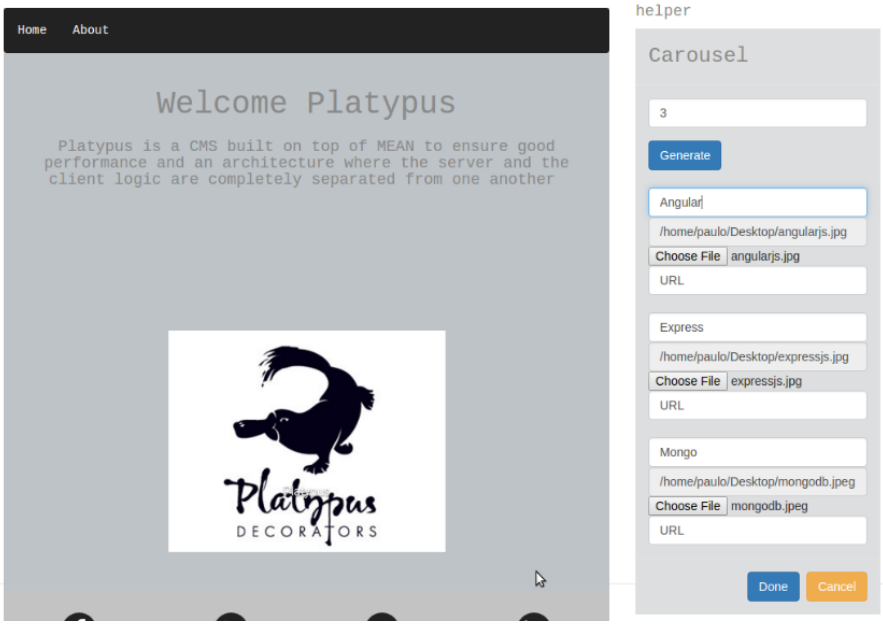

Figura 45: Agregando Carousel [Figura]. "Elaboración Propia"

Luego se agregó un tanto más contenido, algunos iconos con un texto descriptivo y un acordeón en la parte inferior, en la figura 49 podemos apreciar el resultado de los pasos descritos.

Figura 46: Resultado de agregación de Contenido [Figura]. "Elaboración Propia"

Ya habiendo elaborado una página Web sencilla, se guardaran los cambios y llevaremos la página web a los ambientes de desarrollo y producción (figura 50).

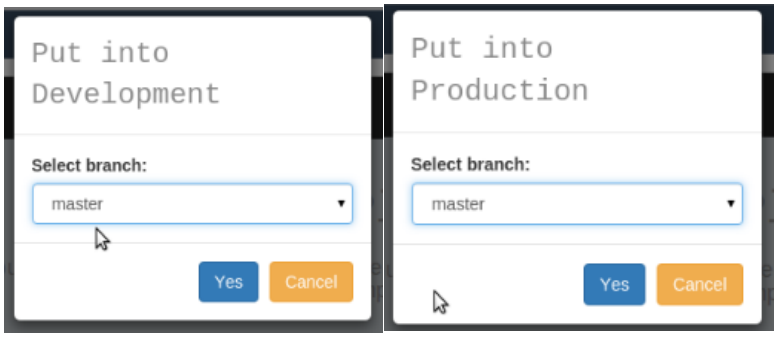

Figura 47: Llevando el Proyecto a Desarrollo y Producción [Figura]. "Elaboración Propia"

Una vez el proyecto se encuentra en Producción este está disponible en la web, específicamente en el URL: http://platypus.inworknet.com:81/

Conclusiones y Recomendaciones

5.1 Conclusiones

Al finalizar el presente Trabajo Especial de Grado, los objetivos propuestos al inicio de la investigación fueron alcanzados con éxito. Este nuevo Sistema Manejador de Contenido aunque con una cantidad reducida de funcionalidades, provee de todo lo necesario para el futuro crecimiento y extensión de la herramienta.

Una ligera modificación de la metodología ágil SCRUM fue utilizada para realizar los diferentes componentes que comprenden este nuevo CMS. Primeramente fue realizado un levantamiento de requerimientos donde se captaron todas la funcionalidades necesarias para esta primera versión, luego se procedió a diseñar una solución que satisficiera estos requerimientos, mediante el uso de distintos artefactos para el desarrollo bien estructurado y documentado de software.

Luego se procedió a desarrollar la aplicación de entrega de contenido de este CMS, buscando e integrando todas las tecnologías que cumplieran con los requerimientos de este módulo, para luego realizar la documentación de los distintos servicios que lo componen. Dicha documentación puede ser observada en las siguientes direcciones URL:

- Documentación del Módulo Platypus API (Consultado 08/05/16):
  - http://platydev.inworknet.com:3005/api
- Documentación del Módulo Platypus App's Host (Consultado 08/05/16):
  - http://platyprod.inworknet.com:3006/api

Por último se llevó a cabo el desarrollo de la aplicación de manejo de contenido, esta fue el mayor reto dentro del TEG para estudiante desarrollador, primero se llevó acabo la integración de los servicios de la aplicación de entrega de contenido para luego desarrollar distintos servicios y directivas utilizados para lograr el funcionamiento de deseado de la aplicación de manejo de contenido.

En una perspectiva general el producto de Software logrado, parece tener un conjunto muy reducido de funcionalidades, sin embargo dichas funcionalidades forman una buena base para el futuro crecimiento de la herramienta.

5.2 Recomendaciones

- Desarrollar funcionalidades dentro de Aplicación de Entrega de contenido para que esta:
    - Tenga la capacidad para manipular el servidor de DNS asociado a esta, para así poder agregar nuevos dominios o subdominios.
    - Brindar funcionalidades para la exportación de proyectos que se encuentran en repositorios ajenos a la herramienta.

- Para mejorar y extender el funcionamiento de aplicación de manejo de contenido se plantea la extensión del servicio *Template Service* para brindar más opciones de contenido y funcionalidades a los usuarios de la herramienta, algunas de estas extensiones podrían ser:
    - Funcionalidad para el desarrollo de Web APIs de manera local.
    - Funcionalidad la exportación de estos en ambientes de desarrollo y producción.

Anexos

6.1 Diagrama de Casos de Uso

Actor de Casos de Uso	Roles representados en el Sistema
Usuario	Todos los Roles
Administrador	Administrador

Tabla 3: Leyenda de relación de actores de casos de uso con respecto a roles del sistema. [Tabla] "Elaboración Propia"

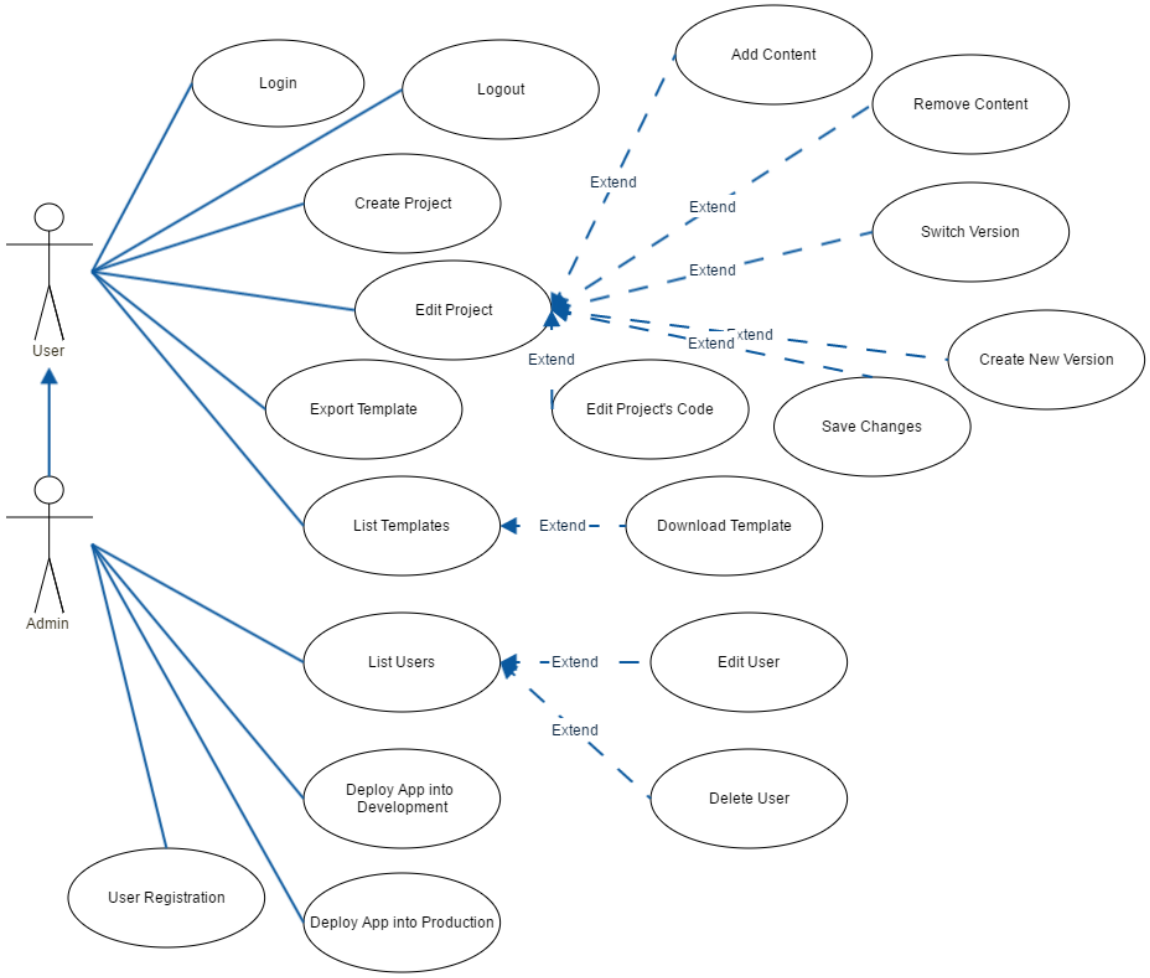

Figura 48. Diagrama de Casos de Uso [Figura]. "Elaboración Propia"

6.1.1 Especificación de Casos de Uso

Use Case	User Registration
Actor	Administrator
Description	The Administrator provides a username, password, email and userType to create new users
Precondition	-
Postcondition	The created user can now access the application
Basic Flow	1. Fill the corresponding fields 2. Submit the Form
Alternate Flow	1. Fill the corresponding fields 2. Submit the Form 3. Display error message, postcondition is not reached
Notes	-

Use Case	Login
Actor	User / Administrator
Description	The actor provides username and password to be logged into the system
Precondition	the actor must be registered
Postcondition	the actor is logged into the system
1. Provide the username and password in the corresponding fields	1. Provide the username and password in the corresponding fields 2. Submit the Form
Alternate Flow	1. Provide the username and password in the corresponding fields 2. Submit the Form 3. An error message is displayed
Notes	-

Use Case	Logout
Actor	User / Administrator
Description	The actor logout from the system
Precondition	The actor must be logged-in
Postcondition	The actor is no longer logged-in
Basic Flow	1. Clicks Logout/Exit button 2. The login View is shown
Alternate Flow	
Notes	

Use Case	Create Project
Actor	User / Administrator
Description	The Actor presses the “Create Project” button on the Dashboard
Precondition	The actor must be logged-in
Postcondition	A new project is shown in dashboard View
Basic Flow	1 Clicks the Create Project button 2 Fill the forms 3 Submit the Form
Alternate Flow	
Notes	

Use Case	Edit Project
Actor	User / Administrator
Description	The Actor presses the edit button of any of the visible projects
Precondition	The actor must be logged-in
Postcondition	The actor is now on the Editor views
Basic Flow	1 Clicks the Edit Project button 2 The editor View is shown
Alternate Flow	
Notes	

Use Case	Add Content
Actor	User / Administrator
Description	The Actor drags and drops some of the listed content
Precondition	The actor must be logged-in and in the Editor view
Postcondition	Content is now added on the Project
Basic Flow	1 Drag and Drop the content 2 Fill out the necessary forms for the generation of content
Alternate Flow	
Notes	

Use Case	Remove Content
Actor	User / Administrator
Description	The Actor drags and drops some of the listed content out the layout view
Precondition	The actor must be logged-in and in the Editor view
Postcondition	Content is now remove from the Project
Basic Flow	1 Drag and Drop the content 2 The Content dissapears
Alternate Flow	-
Notes	-

Use Case	Create New Version
Actor	User / Administrator
Description	The Actor presses the button on the toolbar to create a new version
Precondition	The actor must be logged-in and in the Editor view
Postcondition	a new version is created
Basic Flow	1 The actor presses the button 2 Fill out the necessary forms for the creation of the new version
Alternate Flow	-
Notes	-

Use Case	Change Version
Actor	User / Administrator
Description	The Actor presses the button on the toolbar to use another version
Precondition	The actor must be logged-in and in the Editor view
Postcondition	The version being use is changed
Basic Flow	1 The actor presses the button 2 The system changes the version and shows that versions preview
Alternate Flow	-
Notes	-

Use Case	Save Changes
Actor	User / Administrator
Description	The Actor presses the button on the toolbar to save the different changes done to the project
Precondition	The actor must be logged-in and in the Editor view
Postcondition	The changes are saved
Basic Flow	1 The actor presses the button 2 Loading modal is shown 3 Message Modal Shows with the message: Changes were saved
Alternate Flow	-
Notes	-

Use Case	Edit Projects Code
Actor	User / Administrator
Description	The Actor presses the button on the toolbar to Display the Code Editor and make changes
Precondition	The actor must be logged-in and in the Editor view
Postcondition	The changes are shown on the Projects preview
Basic Flow	1 The actor presses the button 2 Code Editor is Shown 3 The actor modifies the code 4 The changes are shown

Use Case	Edit Projects Code
Alternate Flow	-
Notes	-

Use Case	List Templates
Actor	User / Administrator
Description	The actor selects the option "market"
Precondition	The actor must be logged-in
Postcondition	Market list is Shown
Basic Flow	1 The actor presses the “market” button 2 Loading modal is shown 3 Market template list is shown
Alternate Flow	-
Notes	-

Use Case	Download Templates
Actor	User / Administrator
Description	The actor downloads the selected template
Precondition	The actor must be logged-in and in the market view
Postcondition	A new template is available for usage
Basic Flow	1 The actor presses the “download” button of an specific template 2 Loading modal is shown 3 Success modal is shown 4 Market template list is shown
Alternate Flow	-
Notes	-

Use Case	List User
Actor	Administrator
Description	The actor click the “Users” option in the navbar
Precondition	The actor must be logged-in
Postcondition	The User list view is shown

Use Case	List User
Basic Flow	1 The actor presses the “Users” button 2 Loading modal is shown 3 User list view is shown
Alternate Flow	-
Notes	-

Use Case	Edit User
Actor	Administrator
Description	The actor edits the selected user
Precondition	The actor must be logged-in and in the user list view
Postcondition	Changes are applied to the selected user
Basic Flow	1 The actor presses the “edit” button of an specific user 2 Loading modal is shown 3 Fill the forms 4 Submit the forms 5 Changes are applied and shown
Alternate Flow	-
Notes	-

Use Case	Delete User
Actor	Administrator
Description	The actor deletes the selected user
Precondition	The actor must be logged-in and in the user list view
Postcondition	The selected user is deleted
Basic Flow	1 The actor presses the “delete” button of an specific user 2 Loading modal is shown 3 Changes are applied and shown
Alternate Flow	-
Notes	-

Use Case	Deploy App Into Development
Actor	User/Administrator
Description	The Actor presses the button on the toolbar to put the Current project into develop enviroment
Precondition	The actor must be logged-in and in the Editor view
Postcondition	The Project is now on Develop enviroment
Basic Flow	1 The actor presses the “Put into development” button 2 Loading modal is shown 3 Changes are applied and shown
Alternate Flow	-
Notes	-

Use Case	Deploy App Into Production
Actor	Administrator
Description	The Actor presses the button on the toolbar to put the Current project into Production enviroment
Precondition	The actor must be logged-in and in the Editor view and the Project must be into develop environment already
Postcondition	The Project is now on Production enviroment
Basic Flow	1 The actor presses the “Put into Production” button 2 Loading modal is shown 3 Changes are applied and shown 4 The project is now available on the web
Alternate Flow	-
Notes	-

6.2 Diagrama de Objetos del Dominio

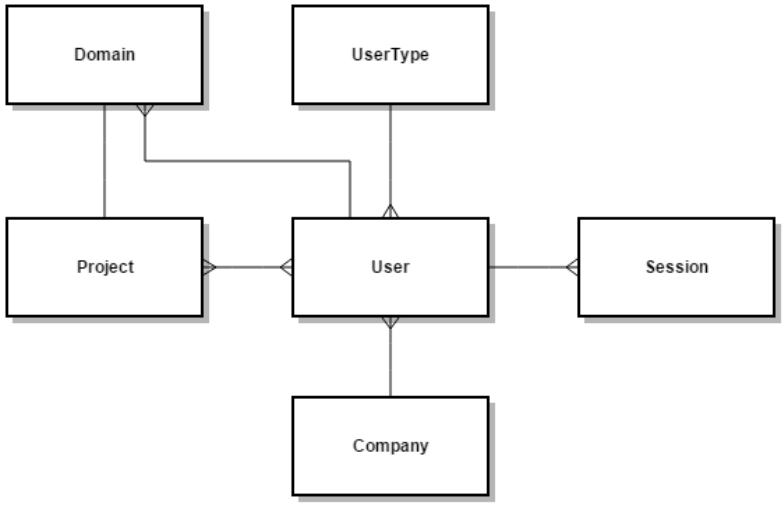

Figura 49: Diagrama de Objetos del Dominio [Figura]. "Elaboración Propia"

6.3 Diagrama de Despliegue

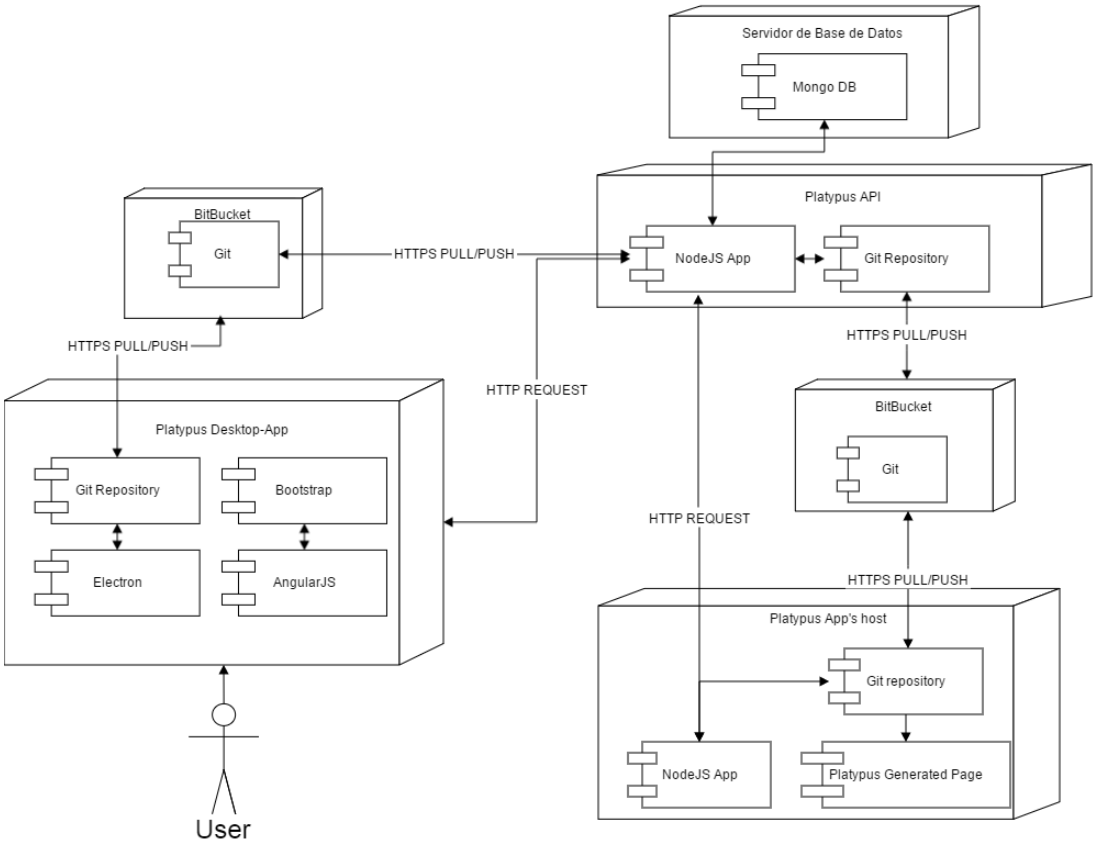

Figura 50: Diagrama de Despliegue [Figura]. "Elaboración Propia"

Referencias

[1] WPBeginner. (2016) News, The History of Wordpress, Fecha de Consulta: 09 de Abril 2016, URL: http://www.wpbeginner.com/news/the-history-of-wordpress/

[2] W3techs. (2016) Content Management, Usage of the Content management System for Websites, Fecha de Consulta: 05 de Abril 2016, URL: http://w3techs.com/technologies/overview/content_management/all

[3] Archive. (2016) Interview With Stefan Esser, Fecha de Consulta: 05 de Abril 2016, URL: http://web.archive.org/web/20121013080700/http://blogsecurity.net/wordpress/interview-280607

[4] HostingAdvice. (2015) Comparing Node.JS vs PHP performance, Fecha de Consulta: 06 de Abril 2016, URL: http://www.hostingadvice.com/blog/comparing-node-js-vs-php-performance/

[5] Websitesetup. (2016) Comparison, WORDPRESS VS DRUPAL VS JOOMLA + CMS "Comparison Chart", Fecha de Consulta: 05 de Abril 2016, URL: http://websitesetup.org/cms-comparison-wordpress-vs-joomla-drupal/

[6] Geerling J. (2015). BenchMarking PHP 7 vs HHVM with Drupal and Wordpress, Fecha de Consulta: 06 de Abril 2016, URL: http://www.jeffgeerling.com/blogs/jeff-geerling/benchmarking-drupal-8-php-7-vs-hhvm

[7] Yotta. (2016). Yotta Company, About, Fecha de Consulta: 06 de Abril 2016, URL: http://www.yottaa.com/company/about/

[8] Yotta. (2016). Application Optimization, BenchMarking Performance of 8 CMS platforms, Fecha de Consulta: 06 de Abril 2016, URL: http://www.yottaa.com/company/blog/application-optimization/benchmarking-performance-of-8-cms-platforms-who-is-slowest/

[9] Techtarget. (2016) Content Management, Content Management Systems, Fecha de Consulta: 05 de Abril 2016, URL: http://searchsoa.techtarget.com/definition/content-management-system

[10] S. Luján Mora. Programación En Internet: Clientes Web. ECU. Octubre, 2001

[11] C. Mellon. Software Engeenering Institude. Client/Server Software Architectures: An Overview, Carnegie Mellon University. Septiembre, 2003

[12] G. Krasner, S. Pope. A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System. ParcPlace Systems, Inc. 1998

[13] W3C. (2016) Breve descripcion de los Servicios web, Guía breve sobre servicios Web, 1, Fecha de Consulta: 05 de Abril 2016, URL: http://www.w3c.es/Divulgacion/GuiasBreves/ServiciosWeb

[14] P. Blanco, R. Kotermanski, P. Merson. Evaluating a Service-Oriented Architecture. Software Architecture Technology Initiative. Septiembre 2007.

[15] HiperTextual. (2016) ¿Qué es una API? , ¿Qué es una API?, Fecha de Consulta: 05 de Abril 2016, URL: http://hipertextual.com/archivo/2014/05/que-es-api/

[16] MWaySolutions. (2015) Client vs Server Side Rendering, Fecha de Consulta: 06 de Abril 2016, URL: http://blog.mwaysolutions.com/2013/11/08/client-vs-serverside-rendering-the-big-battle-2/

[17] PHP. (2016) PHP, Documentation, Fecha de Consulta: 05 de Abril 2016, URL: http://php.net/docs.php

[18] HHVM. (2016) HHVM Documentation, What is HHVM? , Fecha de Consulta: 06 de Abril 2016, URL: https://docs.hhvm.com/hhvm/

[19] Facebook. (2016) Facebook Engineering, The HipHop Virtual Machine, Fecha de Consulta: 06 de Abril 2016, URL: https://www.facebook.com/notes/facebook-engineering/the-hiphop-virtual-machine/10150415177928920

[20] Facebook. (2016) HHVM License, Fecha de Consulta: 06 de Abril 2016, URL: https://github.com/facebook/hhvm#license

[21] JSTheRightWay. (2016) JavaScript, Getting Started, Fecha de Consulta: 05 de Abril 2016, URL: http://jstherightway.org/#getting-started

[22] Tutorials Point. (2016) Node.JS, Node.JS Introduction, Fecha de Consulta: 05 de Abril 2016, URL: http://www.tutorialspoint.com/nodejs/nodejs_introduction.htm

[23] NPMjs. (2016) Getting Started, What is NPM?, Fecha de Consulta: 05 de Abril 2016, URL: https://docs.npmjs.com/getting-started/what-is-npm

[24] ExpressJS. (2016) Resources, Glossary, Fecha de Consulta: 05 de Abril 2016, URL: http://expressjs.com/en/resources/glossary.html

[25] AngularJS. (2016) Guide, Introduction, Fecha de Consulta: 05 de Abril 2016, URL: https://docs.angularjs.org/guide/introduction

[26] ElectronJS. (2016) Electron Documentation, Quick Start, Fecha de Consulta: 05 de Abril 2016, URL: http://electron.atom.io/docs/v0.36.5/tutorial/quick-start/

[27] Twitter BootStrap. (2016), Bootstrap from Twitter, Fecha de Consulta: 09 de Abril 2016, URL: https://blog.twitter.com/2011/bootstrap-from-twitter

[28] Git-Scm. (2016) Getting started, Git Basics, Fecha de Consulta: 05 de Abril 2016, URL: https://git-scm.com/book/en/v2/Getting-Started-Git-Basics

[29] MongoDB. (2016) Manual, Getting Started, Fecha de Consulta: 05 de Abril 2016, URL: https://docs.mongodb.org/manual

[30] SCRUMstudy. A Guide to the SCRUM BODY OF KNOWLEDGE, Library of Congress Cataloging in Publication Data, 2013.