
**Trabajo Final de Graduación**

**Ingeniería en Software**

**Proyecto de Aplicación Profesional (PAP)**

# Sistema de Administración de Contenido en la nube con integración en clientes web

**Adrián Gallardo**

**SOF00311**

**Fecha: 05/08/2017**

**Docente: Ing. Adriana Pérez**

**Tutor: Ing. Fernando Frías**

## Resumen

A lo largo del presente documento se explica en detalle el dominio del problema, relacionado a los Sistemas de Administración de Contenido Web (CMS Web), a la vez que se profundiza en las distintas tecnologías y procesos involucrados que se deben conocer para poder entender y abordar la problemática.

Como resultado del trabajo llevado a cabo se obtuvo un producto de software orientado a resolver las falencias detectadas en los sistemas analizados y aprovechar las oportunidades de mejora que posibilitan las nuevas tecnologías disponibles en el mercado.

El proyecto se desarrolla en el marco del Trabajo Final de Graduación para la Universidad Empresarial Siglo 21.

**TABLA DE CONTENIDO**

## Historial de cambios

| Fecha | Versión | Descripción |
|---|---|---|
| 01/03/2017 | 1 | Versión Inicial |
| 01/05/2017 | 2 | Corrección basados en rubrica v1<br>Análisis y Diseño |
| 13/06/2017 | 3 | Correcciones basadas en rúbrica v2<br>Modelo de negocio, administración proyecto, análisis riesgos<br>Sprints 1 y 2 |
| 20/06/2017 | 4 | Flujo de fondos |
| 26/07/2017 | 5 | Resumen<br>Sprints 3, 4, 5<br>Conclusiones<br>Revisión y corrección general de redacción<br>Referencias en tablas y figuras |
| 05/08/2017 | 6 | Revisión y corrección general de redacción<br>Ajustes generales en formato del documento |

## Título

Sistema de Administración de Contenido en la nube con integración en clientes web.

## 1. Introducción - Marco de referencia institucional

Un CMS (o Sistemas de Administración de Contenido por sus siglas en inglés) es un sistema informático que les permite a sus usuarios generar, editar y eliminar contenido (textual o multimedia) mediante simples formularios donde se ingresan los datos.

Dicho contenido es almacenado en una base de datos, y luego puede ser procesado y transformado en algún otro formato, como por ejemplo en páginas web.

Estas herramientas buscan ser lo suficientemente simples y flexibles para permitir que cualquier persona pueda gestionarlos, incluso sin poseer conocimientos de programación.

En la actualidad existe un gran número de soluciones de este tipo. Algunas son muy populares y difundidas (como por ejemplo WordPress) y muchas otras apenas conocidas o incluso en desuso.

No obstante el gran número de funciones y virtudes que suelen proveer este tipo de sistemas existen algunos puntos débiles que se buscarán abordar y solucionar a lo largo del presente proyecto enmarcado en el Trabajo Final de Graduación de la carrera de Ingeniería en Software.

Como resultado de este proyecto se obtendrá un sistema CMS que dará solución a dichos inconvenientes con el objetivo de constituirse como una herramienta alternativa competitiva en el mercado actual.

## 2. Antecedentes

En la actualidad existen diferentes productos CMS que pueden ser utilizados para desarrollar sitios web con contenido administrable. El fin principal que persiguen es solucionar la problemática recurrente de permitir a sus usuarios gestionar, por sus propios medios, la información que visualizarán los visitantes de su sitio web.

Muchos de ellos son open source (de código abierto) y gratuitos, lo que permite reducir considerablemente los tiempos y costos de desarrollo.

Algunos sistemas CMS posibilitan un alto grado de personalización no sólo en cuanto a estilos y diseño sino también respecto a las características o funcionalidades que se puedan añadir a través de complementos o extensiones (como por ejemplo incorporar un

carrito de compra). Estas características pueden venir incluidas en el producto o ser adquiridas por separado.

Cuando una herramienta CMS no ofrece todas las funcionalidades requeridas por el usuario entonces se puede optar por modificar y extender el código fuente (siempre y cuando esto sea posible), o bien decidirse por desarrollar e implementar un sistema CMS a medida (lo que acarrea inevitablemente mayores costos y tiempos de implementación).

## 3. Descripción del área problemática

Algunas dificultades que se presentan a la hora de integrar o trabajar con una nueva herramienta CMS son:

- **Dificultad a la hora de configurar e instalar servidores y bases de datos:**

  La tarea de instalar un servidor y configurar una base de datos suele intimidar (y en cierto sentido también excluir) a muchos profesionales ajenos al ámbito de sistemas por falta del conocimiento técnico necesario.

- **Lenta curva de aprendizaje:**

  Se debe aprender a utilizar e integrar la herramienta CMS y, en muchos casos, es necesario además contar con cierto conocimiento y experiencia en el lenguaje de programación en el que fue esta desarrollado el mismo;

- **Limitación en el uso de lenguajes y plataformas:**

  La elección de un software CMS tiene implicancias sobre el lenguaje de programación, plataforma de servidor y motor de base de datos a utilizar puesto que deben ser compatibles. Esto limita las opciones del usuario a la hora de abordar un nuevo proyecto. Además, si se requiere integrar una herramienta CMS en un sistema web ya existente no se podrá llevar a cabo a menos que se cumplan las condiciones de compatibilidad antes mencionadas.

- **Dificultad para personalizar el diseño y apariencia:**

  Muchos sistemas CMS vienen acompañados por una serie de templates o diseños predefinidos que pueden utilizarse como punto de partida para la web del usuario. En algunos casos el sistema provee además de algunas opciones configurables que permiten ajustar la apariencia general de la misma.

  Si bien muchas veces estas opciones de personalización pueden resultar suficientes es imposible que satisfagan las necesidades concretas de todos los usuarios. En casos de requerir un mayor nivel de personalización será necesario modificar dichos templates lo cual puede resultar muy complejo ya que nuevamente será necesario tener conocimientos en lenguajes de programación.

- **Funcionalidades integradas verticalmente:**

  Muchos sistemas CMS están pensados para utilizarse como una solución de software integral. En estos casos las responsabilidades del mismo exceden la simple tarea de administrar contenido (por ejemplo brindando autenticación, carritos de compra o estadísticas). Al no ser soluciones modulares implica que deban integrarse como un todo, lo que no siempre resulta ser una alternativa viable.

## 4. Formulación de la problemática

- ¿Qué características y funcionalidades básicas debe proveer un software CMS?
- ¿Qué procesos pueden cambiarse u optimizarse para facilitar la adopción, instalación y mantenimiento?
- ¿Cuáles tecnologías o arquitecturas pueden utilizarse para lograr que un sistema CMS pueda ser integrado independientemente de la tecnología subyacente en el sistema del usuario?
- ¿Puede desarrollarse una solución que se limite a la administración del contenido y proporcionar características extra de forma modular?
- ¿Qué patrones de arquitectura y diseño pueden aplicarse para separar la capa de datos de la capa lógica y de presentación?

## 5. Justificación

El desarrollo de un sistema CMS que pueda ser integrado en cualquier sitio o sistema web resultará de gran importancia por los siguientes motivos:

- No se condicionará a los usuarios sobre cuál tecnología (lenguaje de programación, servidor o base de datos) utilizar, permitiéndoles elegir aquellas que resulten más apropiadas para sus proyectos o bien aquellas con las que se sientan más cómodos;
- Si un usuario decide cambiar algún componente tecnológico en su sistema podrá seguir utilizando la misma herramienta CMS sin problema;
- El usuario sólo tendrá que aprender a utilizar un único CMS y luego podrá integrarlo en cualquier proyecto web sin importar su naturaleza tecnológica, esto reduce la curva de aprendizaje.

Además, logrando reducir la complejidad técnica a la hora de instalar, configurar, mantener y personalizar el sistema CMS se posibilitará la adopción y uso por parte de usuarios ajenos al ámbito de sistemas (como Diseñadores Gráficos y Web), ampliando así el público objetivo al que apunta este tipo de soluciones de software.

## 6. Objetivo general del proyecto

Establecer un marco teórico y técnico para implementar un Sistema de Administración de Contenido Web (CMS Web) que le permita al usuario gestionar y publicar el contenido de su página web mediante una herramienta online. Dicho contenido luego podrá ser incorporado en cualquier aplicación de origen web sin importar el lenguaje de programación o plataforma de servidor utilizado.

## 7. Objetivos específicos del proyecto

- Comprender la problemática y necesidades de los usuarios de los CMS;
- Establecer un marco teórico que permita comprender el contexto de negocio junto con las tecnologías básicas involucradas en el mismo;
- Seleccionar tecnologías adecuadas que permitan llevar adelante el proyecto;
- Diseñar y desarrollar un producto de software que dé solución a la problemática planteada;
- Demostrar los beneficios que obtendrá el usuario al utilizar el producto CMS propuesto.

## 8. Objetivo general del sistema

- Permitir que usuarios de distintas tecnologías web puedan utilizar un único sistema CMS para administrar el contenido de sus webs.
- Facilitar el desarrollo y mantenimiento de los templates de las páginas al separar la capa de datos de la capa de lógica y presentación.
- Optimizar la performance y tiempo de respuesta del servidor al encargarse sólo de la tarea de servir el contenido, delegando la responsabilidad de renderizar las páginas a los clientes web con los que se conecte.

## 9. Límite

Comprende los procesos necesarios desde que el usuario del CMS carga el contenido hasta que este es servido y mostrado en su página web cuando un visitante accede a la misma.

# 10. Alcance

El sistema contempla las funcionalidades que le permitirán a un usuario registrarse y acceder al sistema para luego poder gestionar de manera online los datos de sus aplicaciones y sus respectivos contenidos. Además, como parte de la solución, se desarrollara una librería que podrá ser utilizada en cualquier cliente web para permitir conectarse al CMS e integrar los contenidos del usuario en su propia página web.

# 11. Marco Teórico

## 11.1. Actividad del cliente

Si bien es un proyecto personal, resulta imprescindible conocer la naturaleza de la problemática que se pretende abordar. Para lograr esto se necesita conocer qué es un CMS, sus características, enfoques, usos, usuarios y principales soluciones de software de este tipo existentes en el mercado.

## 11.2. CMS

Según el Centro de Apoyo Tecnológico a Emprendedores (2012), un Sistema de Gestión de Contenido o CMS (por sus siglas en inglés) es un término genérico que se utiliza para referenciar a un tipo de software cuya finalidad es permitir que el usuario administre información (ya sea en formato textual o multimedia), existiendo al menos cuatro grandes subtipos:

- Administradores de contenidos empresariales (ECM);
- Administradores de contenidos web (WCM);
- Administradores de documentos y/o contenidos multimedia (DMS);
- Administradores de contenido para el aprendizaje (LCMS)

No obstante esta diferenciación técnica, el acrónimo CMS se utiliza por lo general para hacer referencia a los administradores de contenidos web (Centro de Apoyo Tecnológico a Emprendedores, 2012). Ya que esta es la acepción más extendida y aceptada en el mercado es la que se utilizará en el marco del presente proyecto para hacer referencia a este tipo de soluciones.

Un software CMS le permite al usuario crear, modificar o eliminar contenido de una manera simple e intuitiva mediante el uso de formularios web. Luego, este contenido puede ser publicado en formato de página HTML para ser accedido a través de internet.

Además de estas funcionalidades básicas, hoy en día es común encontrar administradores de contenido que ofrecen un gran número de prestaciones adicionales. A modo orientativo, el Centro de Apoyo Tecnológico a Emprendedores (2012) ha enumerado algunas de estas características clasificándolas en 3 escenarios que varían de acuerdo a la complejidad y tamaño del sitio y a la magnitud de la organización.

A continuación se describen dichos escenarios.

- **Portales simples:** en este grupo se engloban las páginas webs cuyo propósito es el de tener presencia básica en internet.

Las funcionalidades más solicitadas son:

  - Gestión de noticias;
  - Gestión de banners;
  - Gestión de galería de imágenes;
  - Formularios de contacto;
  - Sindicación de contenido (RSS, Atom);
  - Editores de texto enriquecido (WYSIWYG);
  - Integración con redes sociales;
  - Generación de blogs y foros;
  - Carritos de compra;
  - Herramientas de optimización en buscadores (SEO);
  - Estadísticas de usuarios;
  - URLs amigables.

- **Portales complejos:** este es un escenario más exigente donde se requiere un mayor nivel de personalización en cuanto a la estética y formato de presentación. Debe permitir y facilitar la gestión de contenido por parte de múltiples usuarios.

Funcionalidades deseadas en este contexto:

  - Personalización del contenido;
  - Separación de presentación y contenido (MVC);

- Gestión de taxonomías;
- Gestión avanzada de cache;
- Herramientas de traducción automática;
- Versionado de contenido;
- Copias de seguridad;
- Estadísticas complejas;
- Publicación de contenidos por parte de usuarios externos (gestión de diferentes roles).

- **Intranets:** sumado a todo lo anterior, en este escenario se requiere una mayor integración a la forma de trabajo y procesos de la organización.

Para esto se requiere que el CMS ofrezca lo siguiente:

- Gestión flexible de roles y permisos;
- Flexibilidad para integrar con diferentes sistemas y bases de datos;
- Protocolos de autenticación avanzados;
- Cifrado de comunicaciones;
- Disponibilidad de API para integración;
- Búsqueda de texto avanzada;
- Estructuración de contenido personalizable;
- Wiki;
- Chat;
- Gestión de incidencias y atención al usuario;
- Gestión de flujos de trabajo;
- Herramientas colaborativas en línea.

Cabe destacar que la mayoría de los CMS existentes en el mercado son de código abierto e incluyen licencias de uso gratuitas aunque dependiendo de la solución elegida puede ser que sea necesario pagar para obtener alguna funcionalidad extra.

Según el Centro de Apoyo Tecnológico a Emprendedores (2012) estas soluciones de software libre han alcanzado un nivel de madurez y calidad para nada

despreciable y tienen poco o nada que envidiarle a otras soluciones privativas (no libres) de licencias pagas.

## 11.3. Arquitectura de los CMS

> Un Sistema de Gestión de Contenidos (…) es un programa que permite crear una estructura de soporte para la creación, edición, gestión, publicación y administración de contenido digital en diversos formatos. Generalmente los CMS trabajan contra una base de datos, de modo que el editor simplemente la actualiza incluyendo nueva información o editando la existente (Valdespino Alberti, y otros, 2014, pág. 976).

En esta definición podemos apreciar que las soluciones CMS hacen uso de motores de bases de datos donde se guarda la información propiamente dicha. Luego esta información es procesada por el programa CMS y mostrada al usuario en un formato predefinido por una plantilla o template que puede ser personalizada por el usuario.

El uso de bases de datos para el almacenamiento de la información permite y facilita una separación entre la presentación y el contenido, facilitando así la creación, mantenimiento y búsqueda. Luego, el contenido de las páginas será generado en forma dinámica a partir de la información almacenada en la base de datos (Trias, de Castro, Lopez-Sanz, & Marcos, 2015).

Arias (2015) define a los sistemas CMS como “una aplicación independiente para crear, gestionar, almacenar y distribuir el contenido de las páginas Web” (pág. 295).

Aquí el autor destaca un concepto importante en cuanto a la distinción entre la herramienta que permite administrar el contenido (CMS) y el propio sitio web donde se presenta la información.

Al respecto, Niño (2010) explica lo siguiente:

> En los gestores de contenidos existen diferentes partes. Normalmente hay una parte pública (Frontend) que puede ver todo el mundo al acceder al gestor y es donde se encuentran los artículos, encuestas, menús, etc. La otra parte es la de administración (Backend) y a ella sólo tienen acceso un reducido grupo de personas (pág. 70).

Esta separación permite dar una mejor organización del proyecto al dividir el producto de software. Por un lado existirá una aplicación destinada a la administración de contenidos, haciendo foco además en la seguridad y asignación de roles, y la otra aplicación hará uso de este contenido y se enfocará principalmente en aspectos como la usabilidad, navegación, búsqueda y visualización de la información.

Aubry (2011) señala que estas herramientas deben estar pensadas y diseñadas para el usuario y no tanto para los desarrolladores. Es decir, deben ser simples e intuitivas para que cualquier persona no experta pueda darle el uso correspondiente.

El mismo autor expone que los CMS involucran el uso de la siguiente tecnología:

- Un servidor HTTP que capture y dirija los pedidos recibidos por los usuarios a través de un cliente web o browser. Por ejemplo Apache, IIS, NodeJS.
- Un motor de bases de datos para almacenar la información. Por ejemplo MySQL, PostgreSQL, MongoDB.
- Un lenguaje de programación que permita acceder a la base de datos y generar las páginas HTML. Por ejemplo PHP, .NET, JavaScript.

## 11.4. Aplicaciones web

Según Henderson (2006) una aplicación web puede considerarse un intermedio entre un sitio web y una aplicación común de escritorio en el sentido que utiliza elementos de ambos.

Un sitio web se compone de una serie de páginas escritas en algún lenguaje de marcado (como por ejemplo HTML). En estas páginas se encuentra embebida toda la información que verá el usuario.

En cambio, en una aplicación web el contenido no se encuentra inserto directamente en las páginas, sino que éstas funcionan más bien como plantillas o templates que luego serán completadas con diferentes datos, los cuales pueden provenir, por ejemplo, desde una base de datos (Henderson, 2006; Tortajada Cordero, 2014).

Por lo tanto podemos decir que las páginas en una aplicación web se arman dinámicamente en respuesta a la petición del usuario. Por ejemplo, si el usuario de una web de automóviles desea buscar autos de una marca X, entonces la aplicación web utilizará la página de resultados de búsqueda como una plantilla en la cual insertará los datos solicitados por el usuario.

## 11.5. Sitios estáticos Vs. Dinámicos

Tortajada Cordero (2014) realiza la siguiente distinción:

> Las páginas web pueden ser estáticas o dinámicas. Cuando el contenido es predeterminado estamos ante una página estática y cuando el contenido se actualiza en función de unos sucesos temporales y/o circunstanciales determinados estaríamos ante una página dinámica.

(….) Cuando hablamos de páginas web dinámicas entendemos que se construyen en el momento en el que la página es visitada por el usuario. Así pues, el contenido de la página web no es fijo, sino que se construye según la interacción que el usuario hace con la página. La información de este tipo de página suele estar almacenada en bases de datos, de las cuales se extrae una parte según las selecciones o acciones llevadas a cabo por la persona que visita la página web.

## 12.T.I.C (Tecnología de la Información y Comunicación)

Esta sección define las tecnologías, conceptos y metodologías que son relevantes para el desarrollo del proyecto, describiendo además aquellas que poseen un potencial significativo para ser implementadas.

### 12.1. HTML

HTML (HyperText Markup Language) es un lenguaje de hipertexto mediante el cual se escriben la mayoría de las páginas web. Se trata de un documento que combina textos, imágenes, enlaces entre páginas y otras características multimedia. Este documento es capaz de ser accedido, interpretado y renderizado por un navegador web (Eguiluz, Introducción a XHTML, 2009).

Dichos documentos se componen de una serie de etiquetas predefinidas que sirven para añadir estructura, jerarquía y comportamiento a la página, pudiendo organizar la información en bloques semánticos y añadiendo enlaces entre las distintas secciones y documentos.

Actualmente HTML se encuentra en la versión 5 del estándar y es utilizado extensivamente para el desarrollo de páginas webs, dejando de lado tecnologías alternativas como Flash o los Applets de Java, los cuáles se valen de extensiones o plugins (ajenos al navegador) para poder presentar el contenido en la página.

Una página web se compone usualmente de un conjunto de documentos HTML, imágenes, hojas de estilo y scripts.

### 12.2. Hojas de estilo (CSS)

Mediante el uso de hojas de estilo CSS (Cascade Style Sheets) es posible ajustar en detalle la apariencia de una página HTML. Cada página puede incluir una o más hojas de estilos. En cada hoja de estilo se definen reglas que permiten establecer el valor de los atributos de los distintos elementos o secciones de una página permitiendo, por ejemplo, establecer el color y tamaño del texto, imagen o color de fondo, márgenes, posicionamiento respecto de los otros elementos, entre otros. Cada regla está definida

por un selector (por ejemplo de ID, clase, elemento, entre otras) y un conjunto de pares atributo-valor, los cuales se aplicarán sobre las selecciones encontradas en el documento (Eguiluz, Introducción a CSS, 2009).

## 12.3. JavaScript

JavaScript es el lenguaje de scripting más difundido y utilizado para el desarrollo web. Fue desarrollado por Netscape para intentar dar solución a la problemática de las aplicaciones webs que cada vez eran más complejas y las velocidades de navegación (en aquel entonces) muy lentas. Este lenguaje, que se ejecuta en la computadora del cliente, permitió reducir el número de llamados al servidor, por ejemplo, cuando se necesitaban validar datos de un formulario (Eguiluz, Introducción a JavaScript, 2009).

Hoy en día es utilizado extensamente para mejorar la experiencia del usuario a través de la incorporación de componentes personalizados (como calendarios para seleccionar fechas o galería de imágenes), efectos y transiciones, actualizaciones de contenido sin recargar la página, entre otras características.

Este lenguaje adquirió tal popularidad que permitió la aparición de un gran número de librerías y frameworks. Actualmente su aplicación trasciende los límites de ejecución en un cliente web pudiendo desarrollarse con el mismo aplicaciones que corren en el servidor, conectarse con bases de datos, generar scripts para automatizar tareas e incluso desarrollar aplicaciones para dispositivos móviles y escritorio.

## 12.4. Protocolo HTTP

El Protocolo de Transferencia de Hipertexto (HTTP, HyperText Transfer Protocol) permite la comunicación entre aplicaciones clientes (por lo general navegadores web) y servidores HTTP.

Cada comunicación es una transacción HTTP en la que el cliente envía al servidor un mensaje (petición o request en inglés) que contiene una cabecera y, opcionalmente, datos. En este encabezado se define, entre otras opciones, el método (GET, POST, PUT, DELETE), la versión del protocolo (HTTP ó HTTPS), el host de destino y la URL relativa del recurso (Andreu, 2011).

Cuando se ingresa una URL en un navegador, este realiza una petición de tipo GET a la URL ingresada. Luego, procesa el contenido recibido (por lo general código HTML) y lo muestra en pantalla en formato de página web.

Cuando en una web completamos y enviamos un formulario los datos en el mismo suelen enviarse a través del método POST. Este método permite que los datos se envíen junto con las otras cabeceras HTTP, evitando así que queden expuestos en la URL.

## 12.5. Servidores Web y servidores de aplicaciones

Groussard (2010, pág. 16) explica que estos dos términos suelen confundirse y utilizarse indistintamente, sin embargo el funcionamiento y propósito de cada uno de estos tipos de servidores es diferente.

> Un servidor Web no es más que un simple servidor de archivos. Los clientes se dirigen a éste mediante el protocolo HTTP para obtener un recurso. Cuando el servidor Web recibe una petición HTTP, extrae el nombre del recurso solicitado, lo busca en el disco y "lo envuelve" dentro de una respuesta HTTP para transmitirlo al cliente. Éste es el único trabajo que puede realizar.
>
> (…) La función de un servidor de aplicaciones es radicalmente distinta ya que los recursos que le son confiados no son simples archivos estáticos (…). Generalmente la petición concierne código ejecutable alojado en el servidor. Contrariamente a lo que haría un servidor Web en la misma situación, no transfiere al cliente el código sino que lo ejecuta y es el resultado de la ejecución de este código lo que se reenvía al cliente.

Esta diferenciación nos permite concluir y hacer la siguiente relación: los sitios web estáticos se relacionan con simples servidores web, mientras que los sitios web dinámicos guardan relación con los servidores de aplicaciones, en donde el contenido al que se accede primero es pre-procesado antes de ser devuelto al navegador y mostrado en pantalla.

Entonces, ya que los sistemas CMS crean las distintas páginas en base a información almacenada en una base de datos, podemos afirmar que los mismos se ejecutan en servidores de aplicaciones.

## 12.6. Servidores HTTP

Entre los servidores webs más utilizados para el desarrollo de aplicaciones se encuentran Apache e Internet Information Services (IIS). Además, en los últimos años viene creciendo con fuerza el uso de Node.js como plataforma web en la nube.

### 12.6.1. Apache

Apache es un servidor web de código abierto y fue desarrollado por el grupo Apache Software Foundation. Existen versiones capaces de ser instaladas en los principales sistemas operativos, como Windows, Linux y Mac, lo que habla de una gran versatilidad. Se compone de distintos módulos los cuales pueden ser habilitados, deshabilitados y/o configurados a discreción, lo que también aporta gran flexibilidad a la hora de implementar la solución (Maciá Pérez, 2008).

PHP es el lenguaje de servidor más comúnmente utilizado en estos servidores, aunque cabe destacar que también da soporte a otros lenguajes como .NET, Python o Ruby.

Entre sus ventajas encontramos:

- Se desarrolla dentro del proyecto HTTP;
- Muy configurable;
- Ampliamente aceptado y comunidad activa;
- Modular;
- Código abierto;
- Multi-plataforma (Windows, Linux, MacOS);
- Extensible;
- Buen nivel de soporte técnico, documentación y ayuda en internet;
- Es gratis.

Desventajas:

- Difícil de administrar;
- Posee formatos de configuración no estándar.

### 12.6.2. Internet Information Services

Internet Information Services (IIS), es un servidor web y un conjunto de servicios para el sistema operativo Windows.

Este servicio convierte a una PC en un servidor web para Internet. Las aplicaciones suelen generarse mediante el lenguaje ASP o ASP.Net de Microsoft (Maciá Pérez, 2008).

Entre sus ventajas encontramos:

- Capacidades de servidor web integrado;
- Es configurable, seguro y administrable en internet;
- Desarrollado por Microsoft;
- Proporciona servicios de FTP, SMTP, NNTP, HTTP, HTTPS;
- Soporte de seguridad SSL y TLS.

Desventajas:

- No es multiplataforma, sólo funciona en Windows;
- Posee vulnerabilidades conocidas;
- No es modular.

### 12.6.3. NodeJS

NodeJS es un entorno de ejecución para JavaScript que permite interpretar dicho lenguaje fuera de las páginas web. La elección de este lenguaje se debe principalmente a su naturaleza asíncrona y orientada a eventos, lo cual permite construir aplicaciones en red escalables. Además, JavaScript es uno de los lenguajes más difundidos y conocidos en el mundo, lo cual ayudó a facilitar su adopción y contribuir a su popularidad (Acerca de Node.js, s.f).

NodeJS es multiplataforma, es decir que puede ser utilizado en sistemas operativos Windows, Mac o Linux de manera indistinta. Provee de librerías para comunicarse con el sistema operativo y realizar todo tipo de tareas.

Uno de los principales usos que se le da es para funcionar como un servidor web. Fue desarrollado pensando principalmente en la escalabilidad y performance para permitir el procesamiento de miles de peticiones concurrentes. Posee un mecanismo de respuesta asíncrono, lo que significa que cada vez que se tiene que efectuar una operación de lectura o escritura (por ejemplo a un archivo en disco o en una base de datos) el programa no tiene que esperar a que la operación termine, sino que prosigue con el resto de la petición o con otras peticiones y cuando la operación iniciada finaliza entonces se retoma con el proceso original y lo continúa (Pasquali, 2013).

Ventajas:

- Asincrónico, orientado a eventos, lo que provee gran performance para atender múltiples peticiones concurrentes;
- Se codifica en lenguaje JavaScript, lenguaje ampliamente adoptado en el entorno web;
- Multiplataforma;
- Software libre, de código abierto, y soportada por la Fundación Linux;
- Posee el ecosistema más grande de librerías de código abierto en el mundo;
- Curva de aprendizaje rápida;

- Desarrollo FullStack (es posible desarrollar desde aplicaciones web, de servidor y hasta de escritorio);

Desventajas:

- La programación orientada a eventos puede resultar compleja para muchos que están acostumbrados a otros lenguajes de servidor en donde la ejecución es lineal y la concurrencia la resuelve el lenguaje internamente mediante procesos multihilo.

Tabla 1
*Comparación de las características de los servidores web*

| | Apache | IIS | NodeJS |
|---|---|---|---|
| **Código abierto** | Sí | No | Sí |
| **Multiplataforma** | Sí | No | Sí |
| **Lenguajes de programación** | Principalmente PHP, pero también soporta Python, Ruby, entre otros. | ASP y ASP.NET | JavaScript |
| **Popularidad y adopción para desarrollo web** | Muy alta | Alta (limitada por la plataforma y lenguaje soportados) | En crecimiento |
| **Facilidad de uso y configuración** | Fácil de instalar, difícil de optimizar para proyectos de gran escala | Fácil de instalar y configurar mediante interfaz gráfica. | Fácil de instalar, difícil de optimizar para proyectos de gran escala |
| **Usos principales** | Servidor web | Servidor web | Servidor web, Websockets, automatización de tareas |
| **Costo** | Gratuito | Pago, con versión gratuita más limitada | Gratuito |
| **Ventaja diferencial** | Gran comunidad y disponibilidad de frameworks de desarrollo y librerías de código abierto. | Facilidad de aprendizaje y configuración | Repositorio de librerías de código abierto más grande del mundo. Manejo asincrónico. Gran performance ante peticiones concurrentes. |

## 12.7. Programación orientada a objetos

Durán, Gutiérrez, y Pimentel (2007) explican lo siguiente:

> Los conceptos de clase y objeto son los más importantes de la programación orientada a objetos. Un objeto es cualquier cosa tangible o intangible que se

pueda imaginar, definida frente al exterior mediante unos atributos y las operaciones que permiten modificar dichos atributos.

Una clase es una plantilla que permite definir un conjunto de objetos.

La creación de un objeto a partir de una clase se denomina instanciación.

La programación orientada a objetos utiliza este paradigma para la construcción de los programas de software. Cada clase permite modelar alguna entidad o proceso ayudando a facilitar su comprensión y reducir su complejidad. Este modelo generado es una abstracción, una versión simplificada de la realidad.

Durán et al. (2007) enuncian las siguientes ventajas en el uso de la programación orientada a objetos:

- **Adaptabilidad**: facilidad de transporte de un sistema a otro;
- **Reusabilidad**: total o parcial de componentes;
- **Mantenibilidad**: modelando la solución del problema permite reducir su complejidad y distribuir funcionalidades y responsabilidades, lo que facilita la tarea de mantenimiento.

## 12.8. Compiladores e intérpretes

Los lenguajes de programación que se utilizan para el desarrollo de aplicaciones web son de alto nivel, es decir, que su sintaxis se asemeja más al lenguaje humano que al código máquina necesario para ejecutarlo.

Por esta razón es que existen los compiladores e intérpretes, que permiten traducir un lenguaje de alto nivel en un lenguaje objeto que posee las instrucciones a ejecutar en una máquina con un hardware específico (Alfonseca Moreno, de la Cruz Echeandía, Ortega de la Puente, & Pulido Cañabate, 2006).

Algunos lenguajes de programación necesitan de un compilador (como por ejemplo .NET o Java) y otros necesitan un intérprete (como PHP o JavaScript). Según Alfonseca Moreno et al. (2006) ambos tipos de lenguaje poseen ventajas y desventajas que se resumen en la siguiente tabla:

Tabla 2

*Características de los lenguajes de programación interpretados y compilados*

| | Lenguajes interpretados | Lenguajes compilados |
|---|---|---|
| **Tipos de datos de las variables** | Permite cambiar el tipo de dato de una variable en tiempo de ejecución. | Se debe especificar el tipo de dato de la variable al momento de declararla. Este no puede cambiar durante la ejecución del programa. |
| **Ejecución de instrucciones dinámicas** | Permite ejecutar una instrucción a partir de una cadena de caracteres o el contenido de una variable. | No es posible. |
| **Gestión de la memoria** | La asignación y liberación de la memoria está a cargo del intérprete. | Dependiendo del lenguaje suele requerir por parte del desarrollador mayores definiciones para la asignación de la memoria de cada variable, así como también su posterior liberación. |
| **Depuración** | Permite pausar la ejecución del programa en tiempo de ejecución, y en muchos casos es posible modificar el código fuente mientras se ejecuta. | Se debe generar una versión especial del código compilado que permita la depuración. Modificar el programa en tiempo de ejecución no es posible. |
| **Rapidez en el desarrollo** | Al ser más flexibles suelen requerir menos tiempo de desarrollo frente a un lenguaje compilado | Al ser más estructurados suelen requerir mayor tiempo de desarrollo por parte del programador. |
| **Velocidad de los programas ejecutables** | El programa fuente debe interpretarse y traducirse a lenguaje máquina cada vez que es ejecutado. Esto supone una mayor sobrecarga y pérdida de tiempo. | El programa fuente se analiza morfológica, sintáctica y semánticamente solo una vez durante la compilación, lo que permite tiempos de ejecución más performantes. |
| **Tamaño del programa objeto** | Suele ser mayor ya que no presenta optimización y suele incluir parte del intérprete en el código objeto. | Menor ya que presenta optimización del código. |
| **Ejemplos de lenguaje** | PHP, JavaScript | ASP.NET, Java |

## 12.9. Lenguajes de programación para el desarrollo web

Para el desarrollo de aplicaciones web es necesario elegir algún lenguaje de programación que permita su ejecución en un servidor web.

Entre los lenguajes más populares y difundidos para este fin encontramos a PHP, ASP.NET y JAVA, a los que se suman otras tecnologías más modernas en constante crecimiento de adopción como Python y NodeJS.

### 12.9.1. PHP

PHP es un lenguaje de programación con una sintaxis similar al lenguaje C, el cual es interpretado por un servidor web Apache y genera código HTML dinámico (Maciá Pérez, 2008).

Entre sus ventajas podemos mencionar:

- Es de libre distribución;
- Es de código abierto, no se requiere la compra de licencias;
- Extensa comunidad de desarrolladores;
- Amplia disponibilidad de librerías de distribución gratuitas para múltiples propósitos;
- Soporte a distintos motores de base de datos, como MySQL, PostgreSQL, Oracle, MS SQL Server, entre otras;
- Fácil de aprender;
- Orientado a objetos;
- Gran variedad de frameworks de desarrollo.

Desventajas:

- Sigue permitiendo programación estructurada (no orientado a objetos) lo que muchas veces se traduce en el uso de malas prácticas y dificultad de mantenimiento;
- No es un lenguaje fuertemente tipado lo que dificulta encontrar errores durante la codificación;
- Es de tipo interpretado por lo que su ejecución resulta más costosa que un lenguaje compilado.

### 12.9.2. ASP.NET

ASP (Active Server Pages) es una tecnología propietaria de Microsoft que permite crear páginas web dinámicas en el servidor (Mora, 2002). En la actualidad esta tecnología ha sido desplazada por su sucesora, ASP.NET, la cual es un framework para la creación de aplicaciones web donde se puede programar en cualquiera de los lenguajes de .NET como C++, C# o VisualBasic (Conesa Caralt, Rius Gavidia, Ceballos Villach, & Gañán Jiménez, 2010).

Entre sus ventajas se encuentran:

- Desarrollado y mantenido por Microsoft;
- Se puede programar en cualquiera de los lenguajes soportados por .NET;
- Orientado a objetos;
- Refuerza la adopción y uso de buenas prácticas para el diseño y arquitectura de la solución;
- Es compilado, lo que permite un mejor rendimiento y encontrar errores durante la codificación;
- Herramientas de depuración de código muy potentes;
- División entre la capa de código y presentación.

Desventajas:

- Para el desarrollo se requiere una PC con Sistema Operativo Windows;
- El IDE Visual Studio requiere la compra de licencias en sus versiones completas;
- Consume muchos recursos del servidor.

### 12.9.3. JAVA

Java posee librerías que permiten generar aplicaciones web mediante el uso de Servlets que son clases diseñadas para responder con contenido dinámico las peticiones de los clientes. Un Servlet provee un punto de acceso común a la aplicación y luego deriva el procesamiento de la petición de acuerdo a la ruta o recurso que se solicita. Este es un patrón que implementan muchos de los frameworks MVC (Modelo, Vista, Controlador) existentes hoy en día, los cuales proveen de una única interfaz de acceso encargada de distribuir el procesamiento de las peticiones entre los distintos controladores (Perry, 2004).

Una aplicación web con Java se compone de clases que implementan la interfaz de los Servlets, páginas HTML, imágenes, contenido multimedia, JavaServer Pages (JSP), archivos de configuración y las librerías propias de Java (Perry, 2004).

Ventajas:

- Multiplataforma, se ejecuta sobre JVM (Java Virtual Machine);
- Orientado a objetos;

- Escalabilidad y mantenibilidad;
- Páginas compiladas, mejora la performance;
- Soluciones estándares e implementación de buenas prácticas;
- Java es un lenguaje programación ampliamente difundido y con muy buen soporte y documentación;
- Herramientas de generación de código.

Desventajas:

- Difícil de aprender, elevada curva de aprendizaje;
- Para sitios simples requiere escribir mucho código en comparación a soluciones como PHP.

### 12.9.4. Python

Python es un lenguaje de programación que viene tomando mucha fuerza en el último tiempo, sobre todo por su gran adopción en entornos académicos.

Es un lenguaje interpretado, orientado a objetos y de alto nivel que resulta muy llamativo por su sintaxis simple y concisa. Además de eso, es multiplataforma y con él se pueden desarrollar desde aplicaciones de escritorio hasta aplicaciones web. Dicha versatilidad es otra de las características que dieron fuerte impulso a este lenguaje, además de que tiene una curva de aprendizaje relativamente rápida, y hoy en día se dispone de una gran cantidad de librerías de código abierto que pueden incorporarse a los proyectos para acelerar el desarrollo (Martelli, 2006).

Ventajas:

- De código abierto;
- Multiplataforma;
- Multi-propósito (escritorio, web, procesos de consola);
- Orientado a objetos;
- Rápida curva de aprendizaje;
- De adopción creciente, gran comunidad de desarrolladores.

Desventajas:

- Al ser interpretado es un poco más lento de procesar que un lenguaje compilado.

### 12.9.5. JavaScript

JavaScript es un lenguaje de programación que originariamente fue creado para utilizarse sólo en el contexto de un navegador web. Así, mediante el uso de este lenguaje, se puede por ejemplo cambiar el contenido de la página en respuesta a algún evento o acción del usuario.

Si bien los navegadores web también permiten ejecutar otros lenguajes de scripting (como por ejemplo Visual Basic Script), lo cierto es que en la actualidad JavaScript se convirtió en el estándar para el desarrollo web. De hecho, a partir de la última versión de HTML (HTML5) este lenguaje es interpretado por defecto a menos que el desarrollador explícitamente indique lo contrario en la etiqueta script (Mozilla Developer Network, 2016).

Tal es la importancia y popularidad de este lenguaje que en los últimos años comenzaron a surgir distintas plataformas que permiten utilizar este lenguaje para codificar programas que se ejecuten fuera del entorno de un navegador.

Este es el caso, por ejemplo, de NodeJS que utiliza JavaScript como lenguaje para, entre otros fines, desarrollar aplicaciones web en el servidor. Por lo tanto esto posibilita que cualquier desarrollador web, al conocer JavaScript, pueda aprender y desarrollar aplicaciones que corran en el servidor.

Ventajas:

- Es el lenguaje estándar en HTML y capaz de ser interpretado por cualquier navegador web;
- Asíncrono y orientado a eventos;
- Interpretado, de asignación dinámica y no tipado, lo que le da gran flexibilidad y agilidad para el desarrollo;
- Multiplataforma;
- Curva de aprendizaje rápida;
- Desarrollo FullStack (es posible desarrollar desde aplicaciones web, de servidor y hasta de escritorio con el mismo lenguaje).

Desventajas:

- La programación orientada a eventos puede resultar compleja para quienes estén acostumbrados a otros lenguajes de programación donde la ejecución es lineal;
- Al ser un lenguaje de asignación dinámica y no tipado puede resultar más difícil depurar errores, a la vez que los entornos de programación no ofrecen buenos mecanismos de autocompletado de código.

- No ofrece todas las bondades de los lenguajes de programación orientados a objetos, como paquetes, clases, herencia, interfaces y polimorfismo (aunque existen patrones posibles de implementar para resolver muchas de estas falencias).

Tabla 3

*Comparación de características de lenguajes de programación para la web*

| | PHP | ASP.NET | Java | Python | JavaScript |
|---|---|---|---|---|---|
| **Código abierto** | Sí | No | Sí | Sí | Sí |
| **Multiplataforma** | Sí | No | Sí (mediante uso de JVM) | Sí | Sí |
| **Orientado a Objetos** | Sí. También permite programación estructurada | Sí | Sí | Sí. También permite programación estructurada | Sí, basada en prototipos. También permite programación estructurada |
| **Popularidad y adopción para desarrollo web** | Muy alta | Alta | Alta | En crecimiento | Muy alta en web, y en crecimiento para aplicaciones en servidor |
| **Curva de aprendizaje** | Corta | Alta | Alta | Corta | Corta |
| **Interpretado o compilado** | Interpretado | Compilado | Compilado | Interpretado | Interpretado |
| **Usos** | Desarrollo web | Desarrollo web | Multi propósito | Multi propósito | Multi propósito |
| **Costo** | Gratuito | Gratuito | Gratuito | Gratuito | Gratuito |
| **Ventaja diferencial** | Gran comunidad y disponibilidad de frameworks de desarrollo y librerías de código abierto. | Estructurado, recomendado para soluciones de nivel empresarial | Estructurado, recomendado para soluciones de nivel empresarial | Facilidad de aprendizaje y simplicidad de escritura de código | Manejo asincrónico. Gran comunidad y gran cantidad de librerías gratuitas. Programación en el navegador y en el servidor |

## 12.10. Bases de datos relacionales

Sobre este tema Korth (2002, pág. 1) escribe lo siguiente:

> Un sistema gestor de bases de datos (SGBD) consiste en una colección de datos interrelacionados y un conjunto de programas para acceder a dichos datos. La colección de datos, normalmente denominada base de datos, contiene información relevante para una empresa. El objetivo principal de un SGBD es proporcionar una forma de almacenar y recuperar la información de una base de datos de manera que sea tanto práctica como eficiente.

Entre las principales ventajas del uso de bases de datos el autor enumera las siguientes:

- Centraliza y disponibiliza la información para toda persona que la requiera y tenga acceso;
- Eliminación de información redundante (duplicada) e inconsistente;
- Permite acceso concurrente por varios usuarios;
- Independencia de datos y tratamiento; esto es que la representación lógica de los datos difiere de la forma en la que el motor la almacena físicamente, lo que le da libertad para, por ejemplo, optimizar las estructuras con el objetivo de ganar performance;
- Restricciones de seguridad a nivel de tablas, datos y operaciones sobre los mismos.

Las bases de datos constituyen la esencia de los CMS. El contenido de cada una de las páginas es almacenada en una o más tablas que luego pueden ser recuperadas y armadas dinámicamente (mediante algún lenguaje de programación).

Para la implementación de aplicaciones web algunos de los DBMS más populares y utilizados son MySQL, MS SQL Server y Oracle. La decisión sobre cuál usar muchas veces se realiza teniendo en cuenta su afinidad y facilidad de integración con el lenguaje de programación elegido. MySQL suele ser utilizado para proyectos que involucran el uso de tecnologías libres, por lo que su uso va de la mano con el lenguaje PHP y los servidores Apache. MS SQL Server, al ser una tecnología de Microsoft suele utilizarse en conjunto con ASP.NET. Oracle suele utilizarse con la tecnología JSP de Java.

## 12.11. Bases de datos NoSQL

NoSQL significa “No Sólo SQL” y es un término que se utiliza para hacer referencia a toda base de datos de naturaleza no relacional. En este tipo de bases de datos se prioriza un almacenamiento y/o acceso más ágil a la información, sin preocuparse por

mantener integridad ni eliminar duplicidades. Es decir, se trabaja con un modelo desnormalizado y la estructura de la información por lo general es dinámica pudiendo distintas entradas de una misma colección poseer más o menos datos que otras (Ruano Vázquez, 2014).

Resulta importante destacar la posibilidad de almacenar estructuras dinámicas. Esto, en relación a los CMS, puede considerarse una característica muy beneficiosa en términos de flexibilidad pero también en cuanto a performance, ya que la estructura se guarda y recupera tal y como fue definida por el usuario, lo que evita operaciones de unión entre tablas y post procesamiento para poder recuperar la información en el formato requerido.

Respecto de la forma de almacenamiento Bas Abad (2015), en un estudio que realizó sobre distintas tecnologías de bases de datos no relacionales, distingue 4 formatos principales: clave-valor (similar a una tabla hash), almacenamiento en documentos, grafos y columnas. A continuación se describen sólo los 2 primeros tipos mencionados por su potencial uso en el presente proyecto.

**Clave-valor:** el almacenamiento funciona como una Tabla Hash, es decir una estructura dinámica que vincula un dato de entrada (utilizado como clave) con un dato de salida (utilizado como valor). Este tipo de estructuras suele almacenarse en memoria RAM y por lo tanto provee tiempos de lectura y escritura muy rápidos. Ya que el almacenamiento no se realiza en el disco duro sino en la memoria principal, sólo puede ser utilizada en los casos en los que no se requiere guardar gran cantidad de datos. Además, su utilidad depende de que conozcamos las claves para acceder a los valores. Estas estructuras no están optimizadas para resolver consultas ni uniones de datos (Bas Abad, 2015).

Como ejemplo de este tipo de bases de datos podemos mencionar el motor Redis.

**Orientadas a documentos:** éstas almacenan la información como documentos estructurados, generalmente en formato JSON (JavaScript Object Notation). Cada documento es identificado mediante una clave única autogenerada, similar a lo que ocurre con una base de datos relacional. La diferencia con los motores SQL es que aquí la información se guarda en archivos de texto y los datos pueden tener distintos niveles de anidamiento, sin necesidad de estar divididos en tablas (Bas Abad, 2015).

El mismo autor comenta que muchos motores de bases de datos que utilizan esta forma de almacenamiento permiten indexación y consultas más o menos complejas sobre los datos, pero que son incapaces de realizar uniones entre documentos. Si bien esto puede resolverse a nivel aplicación (iterando y construyendo la estructura deseada) lo más común es que no resulte necesario ya que las relaciones pueden anidarse. De esta manera, por ejemplo, un documento de tipo factura podría contener el lisado de elementos que la compone anidados a los datos de su cabecera.

MongoDB es la opción más popular en esta categoría, pero también existen otras opciones como CouchDB o Cassandra.

Tabla 4

*Comparación de características de bases de datos orientadas a documentos*

| | MongoDB | CouchDB | Redis |
|---|---|---|---|
| **Formato** | Documentos en formato BSON | Documentos en formato JSON | Clave valor |
| **Almacenamiento** | En disco | En disco | En memoria |
| **Versionado** | No | Sí | No |
| **Mecanismos de replicación** | Sí | Sí | Si |
| **Transacciones** | No | No | Sí |
| **Fragmentación en múltiples BD** | Sí | Sí | No |
| **Costo** | Gratuito | Gratuito | Gratuito |
| **Uso principal** | Para almacenar estructuras indexables (incluso anidadas). Foco en la performance por sobre las funcionalidades | Para manejo de grandes cantidad de datos y donde importa el versionado de los datos almacenados | Cuando se requiere operaciones de lectura y escrituras muy rápidas para consulta de datos simples y recurrentes |

### 12.12. Desarrollo en capas

Consiste en organizar los componentes de un sistema de software agrupándolos según su función.

Un patrón arquitectónico que implementa esta separación en capas es MVC (Model View Controller). Según Fowler (2011) dichas capas tiene las siguientes responsabilidades:

- **Modelo**: contiene los componentes que representan y gestionan datos relacionados con la aplicación o el negocio.
- **Vista**: es la capa de presentación formada por las interfaces. Muestran el estado actual del modelo de datos y ofrecen mecanismos de interacción
- **Controlador**: formado por componentes que reciben las órdenes del usuario y gestionan la aplicación de la lógica de negocio sobre el modelo de datos

La implementación de este tipo de patrones facilita el entendimiento por parte del equipo de desarrollo y facilita la incorporación de nuevos miembros, y además posibilita el crecimiento de la aplicación en su conjunto, es decir, provee escalabilidad.

Esta es una característica imprescindible en un sistema CMS. Al integrar una solución de este tipo esperamos que pueda encargarse de parte o de la totalidad de la capa del

modelo, para dejar la lógica (controlador) y la presentación (vista) en manos de la aplicación a la que se integre.

## 12.13. API

API es el acrónimo para Application Programming Interface, o Interfaz de Programación de Aplicaciones. Define el conjunto de funciones o métodos de un programa que pueden ser utilizados por otros programas o sistemas.

Por lo general se construyen librerías reutilizables que permiten hacer uso de sus funciones para resolver tareas comunes o repetitivas. Una API representa un contrato que nos indica que invocando una función con determinados parámetros de entrada obtendremos el resultado esperado, de manera que no necesitamos conocer cómo resuelve la implementación internamente.

## 12.14. API REST

Una API REST es similar a una API convencional en cuanto a lo mencionado a contratos y funcionalidades. La diferencia radica en que para acceder a estos servicios lo hacemos mediante el estándar HTTP, tal como indica Marqués (2013) en el siguiente párrafo:

> REST nos permite crear servicios y aplicaciones que pueden ser usadas por cualquier dispositivo o cliente que entienda HTTP, por lo que es increíblemente más simple y convencional que otras alternativas que se han usado en los últimos diez años como SOAP y XML-RPC.

El mismo autor menciona que existen algunas reglas o convenciones a la hora de generar las URLs para los servicios REST:

- Los nombres utilizados no deben implicar una acción, por lo tanto debe evitarse usar verbos en ellos;
- Las acciones se definen por medio de los métodos HTTP convencionales, como son GET para obtener información, POST para crear nueva, PUT para actualizar datos existente y DELETE para borrarlos;
- Cada recurso debe estar vinculada a una única URL;
- Podemos hacer referencia a distintos recursos con una misma URL siempre y cuando se use un verbo HTTP diferente;
- Los recursos pueden anidarse de acuerdo a su jerarquía lógica.

## 12.15. Ingeniería en Software

Según Sommerville (2005), "la ingeniería del software es una disciplina de la ingeniería que comprende todos los aspectos de la producción de software desde las etapas iniciales de la especificación del sistema, hasta el mantenimiento de éste después que se utiliza" (pág. 6).

Esto permite adoptar un enfoque sistemático y organizado para encarar un proyecto de software y lograr efectividad y calidad.

Este autor nos habla además de la existencia de un proceso del software definiéndolo como "un conjunto de actividades y resultados asociados que producen un producto de software. Estas actividades son llevadas a cabo por los ingenieros de software" (Sommerville, 2005, pág. 7).

Para llevar a cabo este proceso el autor enumera 4 actividades fundamentales que son comunes y estarán presentes en cualquier tipo de proyecto de software. Estas actividades son:

- **Especificación del software:** donde los clientes e ingenieros definen el software a producir y las restricciones sobre su operación
- **Desarrollo del software:** donde el software se diseña y programa
- **Validación del software:** donde el software se valida para asegurar que es lo que el cliente requiere
- **Evolución del software:** donde el software se modifica para adaptarlo a los cambios requeridos por el cliente y el mercado

## 12.16. LEAN Project Management

Lledó (2012) describe que las filosofías ágiles aplicadas en el desarrollo de Software mantienen gran relación con la filosofía LEAN que se enfoca en los proyectos de producción masiva. El autor nos habla de 5 principios fundamentales:

1- Especificar los elementos o funcionalidades que agregan valor al proyecto. Cualquier actividad que no aporte en este sentido genera costos y deben ser tenidas en cuenta como candidatas para ser removidas, ya que es probable que el cliente no esté dispuesto a pagar por ellas;

2- Identificar las actividades que agregan valor y priorizarlas de acuerdo a su importancia. Para cada una de ellas se debe poder identificar un entregable y un cliente (interno o externo) al cual esté dirigido el resultado del esfuerzo;

3- Permitir que el valor fluya sin interrupciones. Las actividades que generan valor deben llegar al cliente lo más rápido posible. Esto se consigue, principalmente, eliminando tareas innecesarias y construyendo en forma iterativa;

4- Permitir que el cliente participe en la identificación de valor e incluirlo de manera activa en la validación de los requerimientos relevados e implementados;

5- Buscar de manera continua la perfección, tanto del producto o servicio como también de los procesos y metodologías empleadas.

## 12.17. Manifiesto ágil

El Manifiesto Ágil (Beck, y otros, 2001) surgió ante la necesidad de cambiar la forma en la que se concebía la administración y ejecución de proyectos de software frente a los modelos tradicionales implementados en ese entonces por las grandes compañías, y que dada la vertiginosidad con la que crecían los sistemas y las necesidades cambiantes de los clientes requirió cambiar el enfoque para lograr obtener productos de software en tiempo, forma y presupuesto.

Este manifiesto es considerado el iniciador de la filosofía ágil. En el mismo, los autores se enfocaron sobre cuatro pilares fundamentales:

- Privilegiar a los individuos y sus interacciones frente a los procesos y herramientas;
- Perseguir el objetivo de obtener software funcionando correctamente frente a documentación extensa;
- Colaborar a la par con el cliente en el desarrollo del producto en lugar de guiarse por obligaciones contractuales;
- Responder y ser flexibles ante la necesidad de cambio en vez de estar atados a un plan estricto.

Luego, los autores enuncian que estos objetivos pueden alcanzarse mediante la aplicación de los siguientes principios:

- Entregar en forma temprana y continua el software, planificando cada iteración de manera que cada una agregue valor que pueda ser percibido y validado por el cliente;
- Aceptar los cambios que surjan incluso en etapas avanzadas del ciclo de vida de proyecto, ya que se consideran claves para proporcionar ventaja competitiva al cliente;

- Alentar el trabajo conjunto entre los desarrolladores y los responsables del negocio de manera de anticipar tempranamente cualquier tipo de desviaciones que se produzcan para intentar disminuir su impacto;
- Privilegiar la comunicación cara a cara entre los miembros del equipo de desarrollo y entre ellos y el cliente;
- Reflexionar en conjunto (equipo y cliente) luego de cada iteración sobre cómo lograr ser más efectivos para poder ajustar y perfeccionar tanto el sistema como la metodología de trabajo.

### 12.18. Scrum

Scrum es una metodología ágil de trabajo muy difundida y utilizada para gestionar proyectos de Software.

Palacio (2014) menciona en su libro las principales ventajas de su uso:

- Estrategia de desarrollo incremental, en lugar de planificación y ejecución completa del producto;
- Equipos de trabajo autogestionados, en lugar de procesos rigurosos;
- Solapamiento de las fases de desarrollo, en lugar de secuencialidad estricta.

Del libro del mismo autor podemos extraer los siguientes conceptos clave:

- Se comienza con la visión general de lo que se desea obtener, y luego se descompone el trabajo en partes y se las organiza por prioridad;
- Cada ciclo o iteración recibe el nombre de Sprint;
- Un Sprint se mide en días que podemos definir de acuerdo a las necesidades del proyecto y del equipo de trabajo, pero una vez definida su duración debería respetarse a lo largo de la vida del proyecto;
- Al terminar un Sprint se obtiene un incremento sobre la versión del producto, que debe incluir funcionalidades completas capaces de ser presentadas y demostradas al cliente;
- La evolución dentro de un Sprint se monitorea por medio de reuniones diarias en las que participan todo el equipo y se pretende saber qué se hizo, con qué seguir y si existe algún impedimento sobre alguna tarea;
- Un Sprint está compuesto por un conjunto de Historias de Usuario seleccionadas para trabajar en el intervalo de tiempo actual;

- Las Historias de Usuario son definiciones de tareas que se describen en un lenguaje coloquial (no técnico) la funcionalidad que se pretende implementar;
- Una Historia de Usuario suele ser bastante genérica y abarcar un gran número de funcionalidades y roles dentro del equipo;
- Para gestionar con mayor finalidad una historia ésta puede subdividirse en tareas más chicas;
- Las Historias de Usuario deben describir funcionalidades completas que puedan ser demostradas al final de un Sprint;
- Si una Historia de Usuario es demasiado grande para entrar en un Sprint, ésta debe reformularse y descomponerse en historias de menor tamaño. La historia original recibe el nombre de Historia Épica y se vincula a las otras historias con el objetivo de mantener trazabilidad;
- Cada Historia de Usuario define Criterios de Aceptación las cuales describen las condiciones que debe cumplir el producto o cómo debe comportarse. Estos criterios sirven para verificar si la tarea fue resuelta correctamente de acuerdo a las definiciones del cliente;
- El conjunto de Historias de Usuario, las cuales deben incorporar el 100% de funcionalidades del sistema, recibe el nombre de Pila de Producto (o Product Backlog por su nombre en Inglés);
- El conjunto de Historias de Usuario dentro de un Sprint recibe el nombre de Pila de Sprint (o Sprint Backlog por su nombre en inglés).

Las pilas de producto o sprint pueden ser organizadas y gestionadas mediante un tablero Kanban.

## 12.19. Kanban

“El término Kanban aplicado a la gestión ágil de proyectos se refiere a técnicas de representación visual de información para mejorar la eficiencia en la ejecución de tareas de un proyecto” (Palacio, 2014, pág. 78).

Palacio (2014) explica lo siguiente sobre los tableros Kanban:

- Un tablero Kanban puede ser concebido como una pizarra que se divide en varias secciones verticales, cada una de las cuales representa un estado dentro del ciclo de vida de una tarea o historia de usuario;
- Dicho estados pueden ser elegidos libremente, pero es común incluir al menos los siguientes: pendiente, en curso, listo. Algunos estados adicionales podrían ser implementados para facilitar la gestión de trabajo entre varios miembros del

equipo, por ejemplo, pendiente de testing, validado, en revisión de pares, listo para deploy, entre otros.

- Los estados se organizan de izquierda a derecha en orden cronológico y representan el ciclo de vida de una tarea, desde que es seleccionada para trabajar hasta que se completa y da por finalizada;
- La representación en un tablero permite visualizar a simple vista el avance sobre las tareas y el sprint en general;
- Las tareas deberían estar jerarquizadas por prioridad dentro de cada estadio;
- Por lo general, en Scrum, se crea un tablero por cada Sprint;
- Un Sprint puede darse por terminado cuando todas sus tareas se encuentran en la última columna del tablero.

## 12.20. Desarrollo Dirigido por Tests (TDD)

El Desarrollo Dirigido por Tests (*Test Driven Development*) es una técnica de diseño y construcción muy utilizada para el desarrollo de software.

Blé Jurado (2010) nos menciona que esta técnica se centra en tres pilares fundamentales:

- Implementar sólo las funcionalidades que necesita el cliente;
- Diseñar casos de prueba por cada funcionalidad de manera de reducir el riesgo de que los problemas lleguen a entornos productivos;
- Obtener código más modular y reutilizable.

Además, el mismo autor nos acerca los siguientes pasos básicos para implementar TDD:

- Escribir el caso de prueba para el requisito (en este punto va a fallar puesto que aún no se escribió la lógica del programa que se desea probar);
- Desarrollar el código que permita superar con éxito el caso de prueba diseñado;
- Refactorizar el código (de ser necesario) para eliminar duplicidades u optimizar.

Como se puede apreciar el paradigma convencional de diseñar, codificar y luego probar está invertido. Lo primero que se hace es diseñar la prueba para luego escribir el código que permita cumplimentar la misma. Además, desarrollar teniendo en cuenta los casos de prueba influye directamente en el diseño y arquitectura de nuestro sistema.

El beneficio de implementar TDD no es sólo conseguir una buena cobertura de pruebas de nuestro sistema. Blé Jurado (2010) expone, además, las siguientes ventajas:

- Incrementa la calidad del producto;
- Incrementa la confianza que tenemos sobre el software desarrollado, sobre todo cada vez que se inserta alguna modificación;
- Incrementa la confianza en miembros del equipo menos experimentados;
- Mejora la documentación para los miembros del equipo (actuales y futuros);
- Simplifica el código y funcionalidades;
- Fija pautas para el diseño del sistema.

# 13. Competencia

## 13.1. Soluciones CMS más populares

El sitio web cmsmatrix.org, que se encarga de relevar información y funcionalidades sobre los distintos CMS existentes, posee en su base de datos más de 1200 productos de software de este tipo.

No obstante la amplia variedad de opciones, las soluciones más difundidas y utilizadas son apenas unas pocas. Las estadísticas de adopción de estos CMS más populares son muy significativas (W3 Techs, 2016):

- De los 10 millones de sitios relevados han detectado que un 45% de ellos fue construido con alguna herramienta de CMS, es decir, unos 4.5 millones.
- De este total el 59.3% fue creado utilizando WordPress, es decir, más de 2.7 millones.
- En los puestos 2 y 3 se encuentran Joomla y Drupal con el 6.2% y el 4.9% respectivamente.

Estos números son contundentes. Por un lado respecto al gran uso de herramientas de gestión de contenido web, y por otro lado respecto de las soluciones que han ganado más popularidad en el último tiempo.

Los 3 productos de CMS antes mencionados, WordPress, Joomla y Drupal, son los que se tomarán como referencia de análisis de la competencia.

Entre sus características se encuentran los siguientes puntos en común:

- De código abierto y licencias gratuitas;
- Escritos en PHP, con base de datos MySQL como motor principal;
- Diseño modular, extensible mediante la compra y/o descarga gratuita de plugins o complementos;
- Uso de temas y plantillas predefinidos con opciones de personalización;
- Amplia comunidad de desarrolladores;
- Permiten la descarga de código fuente e instalación en servidores propios;
- Permiten crear contenido en línea y mostrarlo en forma de blog o como un sitio web propiamente dicho;
- Panel de administración con acceso mediante usuario y contraseña;
- WordPress y Joomla ofrecen la posibilidad de crear un sitio web básico desde la misma plataforma web.

Para continuar con el análisis sobre cuál CMS elegir es importante conocer el perfil de usuario que se hará cargo de la construcción y mantenimiento, así como también del nivel de personalización que se requiere para el proyecto. Para el análisis mencionaremos los siguientes perfiles (IsYourWeb, 2015):

- **SiteBuilder:** son capaces de instalar y configurar la plataforma y módulos y gestionar el contenido, pero poseen pocos o nulos conocimientos de programación, lo que les imposibilita personalizar aquellas funcionalidades en las que el CMS no provea de una opción configurable.
- **Programadores:** además de poder instalar la plataforma, crear el sitio y gestionar el contenido son capaces de personalizarlo a niveles que van más allá de lo facilitado desde el panel de administración del CMS. Esto lo consiguen modificando el código fuente, desarrollando sus propias extensiones, modificando las plantillas y templates, etc.

Siguiendo con este análisis de perfiles de usuario se rescatan las siguientes afirmaciones:

- WordPress y Joomla está más enfocado a los usuarios de perfil SiteBuilder;
- Drupal, por sus prestaciones, está más enfocado a los programadores;
- A un SiteBuilder le será más fácil crear un sitio web con WordPress y Joomla que con Drupal;
- A un Desarrollador le será fácil crear un sitio con bastantes prestaciones utilizando WordPress o Joomla, pero si requiere ampliar las funcionalidades

básicas y lograr un mayor nivel de personalización le resultaría más conveniente utilizar Drupal.

Para complementar y sostener estos puntos de vista a continuación se presenta un cuadro donde se comparan las características principales de cada producto.

Tabla 5

*Comparación de las características ofrecidas por los sistemas CMS más utilizados*

| Criterio | WordPress | Joomla! | Drupal |
|---|---|---|---|
| **Posicionamiento SEO** | Bien preparado. Es el mejor para pequeñas y medianas empresas. | Muy bien preparado si eres experto | Muy bien preparado si eres experto |
| **Multilenguaje** | Muy bueno (de pago) o bueno (gratis) | Muy bueno (de serie) | Muy bueno (de serie) |
| **Fácil de usar** | Centrado en el usuario | Algo complicado sin conocimientos de programación | Complicado, difícil |
| **Volumen de complementos o extensiones** | Muchísimos (plugins) | Muchos (extensiones) | Muchos (módulos) |
| **Diseño y aspecto** | Bonito, fácil, amigable. | Depende de las habilidades del usuario | Depende de las habilidades del usuario |
| **Tipo de página web** | Diseños básicos y agradables | Sitios complejos orientados al mundo profesional | Sitios complejos orientados al mundo profesional |
| **Público objetivo** | Principiantes o Emprendedores | Diseñadores web | Desarrolladores web |

Fuente: Séculi (2016)

A modo de conclusión, en el cuadro anterior podemos ver que WordPress apunta a un público más bien principiante, con pocos o nulos conocimientos técnicos y que no requieren grandes niveles de personalización.

## 13.2. Soluciones CMS modernas

No obstante la preferencia de la mayoría de usuarios por las soluciones anteriores lo cierto es que constantemente aparecen nuevas soluciones CMS que buscan aprovechar las nuevas tecnologías para cambiar el paradigma acerca de qué es un CMS y cómo administrar el contenido de las páginas web.

A continuación se mostrarán algunos ejemplos relevados que, por su enfoque, se asemejan más a la propuesta de solución a la que se intenta llegar.

**WebPop:**

Funcionalidades principales:

- Permite la edición de archivos HTML, CSS y JS directamente desde el navegador;
- Permite gestionar secciones, artículos y categorías de búsqueda;
- Para dar de alta los contenidos primero se crea una entidad y se le asocian distintos componentes de formulario que permitan luego cargar la información. Por ejemplo si mi entidad es un Auto puedo arrastrar 2 campos de texto para marca y modelo, un selector de color y un selector numérico para indicar el kilometraje;
- Tanto la estructuración del sitio, como la maquetación y gestión de contenidos se hace online desde la web.

**Simpla:**

Permite administrar el contenido de cualquier sitio web incluyendo una librería desarrollada en JavaScript que ellos proveen.

La idea es que una vez que se tenga el sitio maquetado se asocie cada bloque de contenido (texto, imágenes, mapas, etc.) con un identificador único que se utilizará para sincronizar y guardar el contenido en una base de datos en la nube.

La librería permite además acceder en modo edición para actualizar el contenido directamente desde la web del cliente.

Estas características posibilitan el siguiente flujo de trabajo:

- El cliente se provee los textos iniciales y define el contenido y estructura del sitio;
- El diseñador gráfico diseña las distintas pantallas;

- El diseñador web maqueta el sitio y asigna a cada bloque de contenido un identificador único. Además incluye la librería de Simpla;
- Luego el sitio es publicado online y entregado al cliente;
- El cliente puede acceder en cualquier momento en modo edición y actualizar el contenido a mostrar, directamente desde su propia web. Por ejemplo, si quiere cambiar un título simplemente hace doble clic en el mismo, lo cambia, y esto actualizará automáticamente el contenido en la web para los futuros visitantes.

### 13.3. Conclusiones

Como ya se mencionó, estos ejemplos de CMS modernos se recuperaron y citaron debido a que sirven como un punto de referencia y aportan ideas hacia dónde se puede encaminar la propuesta para ofrecer soluciones que se diferencien de los CMS tradicionales.

A continuación se muestra una tabla comparativa entre estas soluciones y lo que se pretende lograr como propuesta de proyecto:

Tabla 6
*Características de productos CMS modernos y diferenciación con producto CMS a desarrollar*

| | **WebPop** | **Simpla** | **CMS deseado** |
|---|---|---|---|
| **Alojamiento** | Backend y Frontend en la nube | Contenido en la nube, Frontend en hosting del usuario | Backend en la nube, Frontend en hosting del usuario |
| **Entorno de desarrollo (IDE)** | Online a través del Backend de WebPop | Offline en PC del usuario | Offline en PC del usuario |
| **Posibilidad de integración del contenido en cualquier web** | No es posible, toda la web es desarrollada desde el Backend de WebPop | Posible de integrar en cualquier sitio, independiente de la tecnología o servidor | Posible de integrar en cualquier sitio, independiente de la tecnología o servidor |
| **Facilidad de actualización de contenido** | Simple, desde editor en Backend en la nube | Muy simple, desde el propio Frontend | Muy simple, desde el propio Frontend o desde Backend en la nube |

| | | | |
|---|---|---|---|
| **Flexibilidad del contenido administrado** | Flexible a través de la configuración de formularios para dar de alta las entidades | Muy flexible. Puedo crear cualquier contenido y luego vincularlo a un identificador para almacenar su valor | Flexible a través de la configuración de formularios para dar de alta las entidades |
| **Uso de plantillas para visualizar contenido** | Sí, se define la estructura de la entidad y se lo vincula a una plantilla | No, el template de cada página es independiente | Sí, se define la estructura de la entidad y se lo vincula a una plantilla |
| **Cómo se integra el contenido en la web** | La página es devuelta por la aplicación con el contenido incrustado. | Mediante librería en JavaScript se incrusta el contenido en la web del cliente | Mediante librería en JavaScript se incrusta el contenido en la web del cliente |

# 14. Diseño Metodológico

## 14.1. Recolección de datos

Las técnicas de recolección de datos a utilizar en este proyecto serán encuestas y entrevistas con posibles usuarios (desarrolladores web y diseñadores gráficos y web) y lectura y análisis de bibliografía sobre TICs tanto en línea como en libros.

## 14.2. Planificación del proyecto

La planificación y ejecución del presente proyecto se descompondrá en 2 grandes grupos, cada uno de los cuáles será planificado y gestionado mediante un tablero Kanban:

- El primero se utilizará para planificar y gestionar el avance sobre proyecto en general y los contenidos del presente documento;
- El segundo tablero será utilizado para planificar y gestionar las tareas vinculadas al desarrollo del producto de software propuesto.

## 14.3. Metodología de trabajo

Se utilizarán conceptos de Scrum y Kanban para planificar las tareas del proyecto y monitorear el avance del mismo.

Estas metodologías se escogieron ya que son lo suficientemente flexibles para adaptarlas incluso a desarrollos pequeños y unipersonales.

## 14.4. Metodología de desarrollo

Se implementará la metodología TDD (desarrollo orientado por pruebas) en las partes del producto que se consideren críticas, principalmente en aquellas áreas sobre las cuales no se tiene un feedback directo sobre el desempeño del producto.

Esta metodología ayudará a prevenir que los errores lleguen al usuario, servirá como guía para el diseño de la solución, facilitará el proceso de verificación e incrementará la calidad y la confianza de las áreas del producto de software en donde se aplique.

## 14.5. Lenguajes de programación

Se utilizará JavaScript para el desarrollo de todo el producto (Backend, Frontend, Servicios). La elección de este lenguaje se debe a lo siguiente:

- Por un lado, es el lenguaje que aceptan nativamente todos los navegadores web;
- Permite el desarrollo tanto del lado del cliente (Frontend) como del lado del servidor (Backend).
- El uso de un único lenguaje para todo el producto reduce la curva de aprendizaje y permite reutilizar código entre las aplicaciones.

Para el desarrollo de la aplicación Backend y de la capa de servicios se utilizará Node.js.

Se utilizará JavaScript para el desarrollo de la aplicación Frontend.

## 14.6. Frameworks de desarrollo

Para la capa de servicios se utilizará Sails.js por su versatilidad a la hora de conectarse con distintos motores de base de datos relacionales y no relacionales. Además su arquitectura se pensó para facilitar la creación y manejo de APIs REST, las cuáles pueden ser autogeneradas a partir de la configuración de los distintos modelos de datos que requiera nuestra aplicación.

Para la aplicación Backend se utilizará el framework Angular. Este framework permite el desarrollo de aplicaciones completas que se ejecutan en el browser del usuario. Permite el uso de patrones como Inyección de Dependencias, MVC y MVVM. El desarrollo está orientado a componentes. Cada componente es una clase

con cierta lógica vinculada a una vista. Las aplicaciones se conforman de un conjunto de componentes que se relacionan y comunican entre sí. Además cuenta con un gran ecosistema de desarrolladores por lo que es posible conseguir componentes de código abierto de gran calidad que pueden utilizare libremente en los proyectos para acelerar el desarrollo. Este framework es desarrollado y mantenido por Google.

Para la librería cliente se utilizará Riot.js. Este framework se eligió principalmente por su peso reducido (lo que resulta un beneficio directo en cuanto a tiempos de carga del producto a desarrollar) y ya que permite el desarrollo orientado a componentes, lo que aportará modularización y posibilidad de integrar componentes desarrollados por terceros. Además, esta librería puede utilizarse en conjunto con otras librerías de desarrollo, como por ejemplo jQuery, un framework muy popular que facilita, entre otras cosas, la manipulación de documentos HTML.

## 14.7. Bases de datos

Se utilizarán bases de datos tanto de tipo relacional como no relacional.

Como motor de base de datos relacional se utilizará MySQL, elegida principalmente por su fácil configuración y administración y porque es una de las bases de datos más difundidas y utilizadas para proyectos de código abierto. Además es uno de los motores que se integran por defecto al framework Sails.js.

Además, dado que en el proyecto se contempla soportar estructuras de datos flexibles y dinámicas, se optó también por el uso de un motor de base de datos de tipo no relacional como lo es MongoDB. Este motor se eligió por su gran popularidad y facilidad de integración con los lenguajes de programación y frameworks elegidos.

## 14.8. Entorno Integrado de Desarrollo (IDE)

Como software para el desarrollo se utilizará el IDE de código abierto Atom.

Este IDE provee distintos plugins que pueden ser añadidos para facilitar el proceso de desarrollo, incluyendo características como autocompletado de código, marcado de errores, herramientas de depuración de código y análisis estático de sintaxis y errores.

## 14.9. Despliegue

La implementación se realizara en los servidores en la nube de Google Cloud. Se escogió este proveedor ya permite escalar la solución en la medida que se requiera y se cobra por uso real que se le da al servidor.

Además, proveen distintos servicios adicionales muy útiles, como replicación de base de datos, backup automático de código, integración con herramientas de versionado para integración continua, estadísticas de uso, entre otros.

### 14.10. Versionado de código

Se utilizará GIT como herramienta de versionado tanto para código como también para documentación, interfaces, diagramas y cualquier otro artefacto que se cree y utilice durante el desarrollo del producto.

Se utilizará una cuenta gratuita en GitLab que permite la gestión tanto de repositorios públicos como privados.

### 14.11. Planificación

La planificación general de este proyecto se realizó mediante un diagrama GANTT:

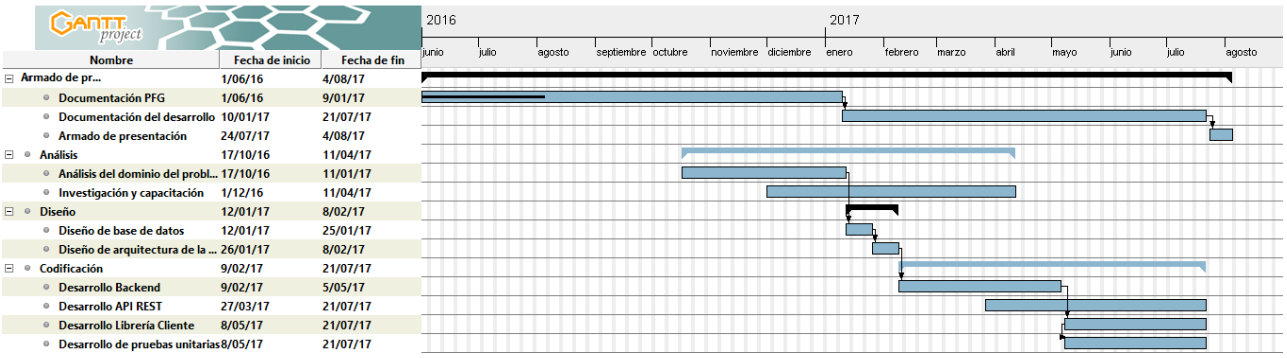


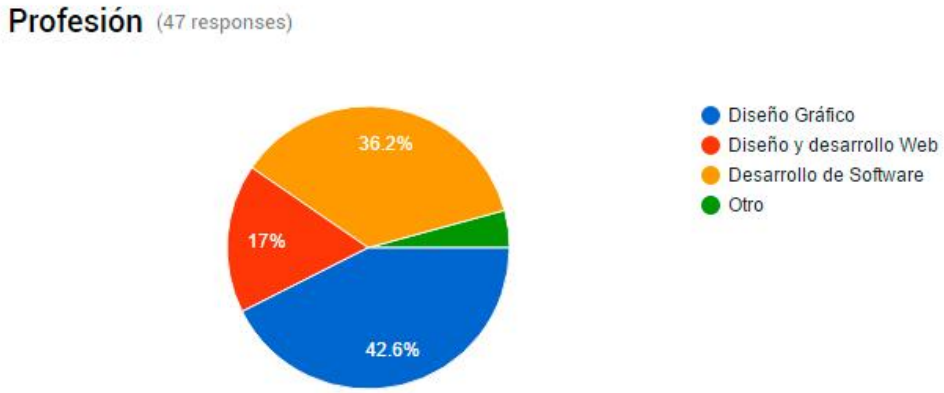

*Figura 1: Diagrama GANTT con la planificación general del proyecto*

## 15. Relevamiento

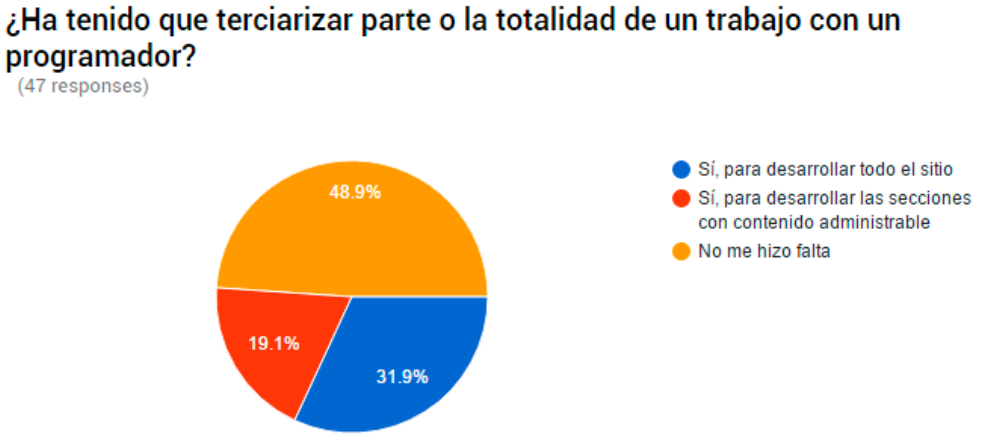
### 15.1. Usuarios

Los potenciales usuarios a los cuáles estará dirigida la solución son principalmente profesionales en las áreas de diseño gráfico y web, a los que sus clientes los contratan para construir un sitio web de pequeño o mediano tamaño.

A continuación se presentan los resultados de una encuesta anónima realizada entre diseñadores gráficos, diseñadores web y programadores.

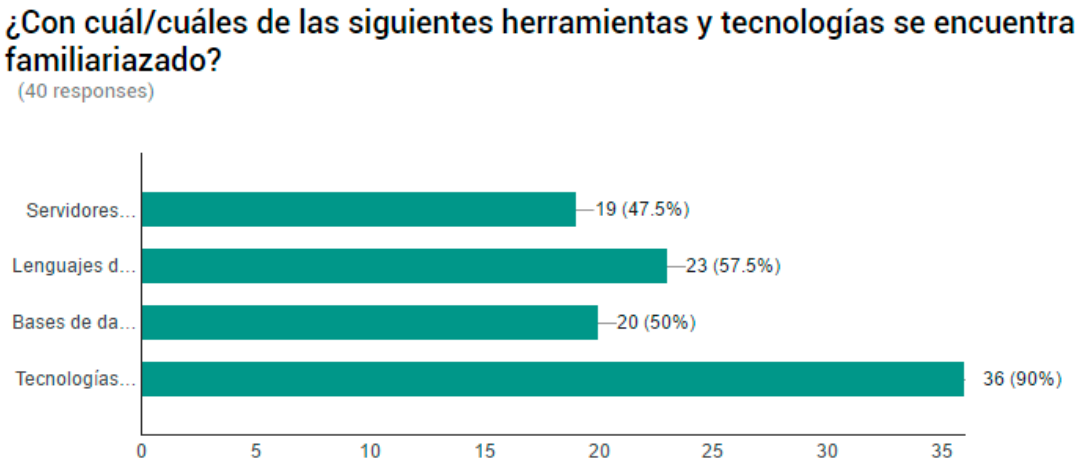



Vemos una distribución pareja entre diseñadores gráficos y programadores, y confirmamos que existen algunos perfiles orientados a las 2 profesiones.




Vemos que la proporción que no necesitó derivar la construcción del sitio web a un tercero se relaciona con la cantidad de programadores y diseñadores con conocimientos de programación.

Además se puede apreciar un número significativo de casos en los cuales el encuestado manifiesta haber tenido la necesidad de terciarizar la construcción del sitio. Podemos suponer que esta cantidad guarda una estrecha relación con la cantidad de diseñadores gráficos encuestados, y la diferencia seguramente se trate de casos en los cuales el diseñador no se haya visto involucrado en trabajos de armados de webs.

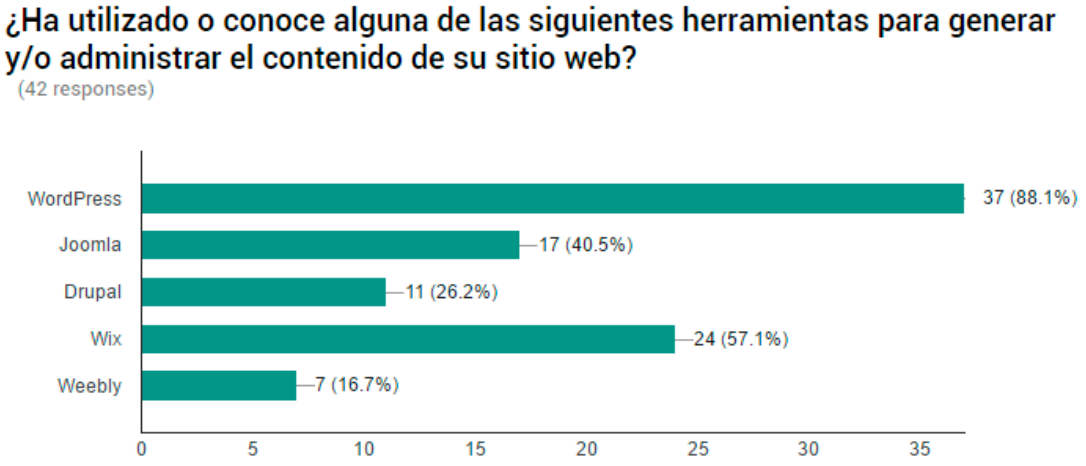



Acá vemos que de las 4 opciones las más desconocidas son:

- Servidores web (Apache, IIS, etc.)
- Lenguajes de programación (PHP, Java, .NET, Phyton, etc.)
- Bases de datos (MySQL, PostgreSQL, SQL Server, Oracle, etc.)

Estas opciones son familiares con aproximadamente el 50% de la población encuestada, lo que se relaciona casi directamente con la cantidad de programadores y diseñadores con experiencia en web.

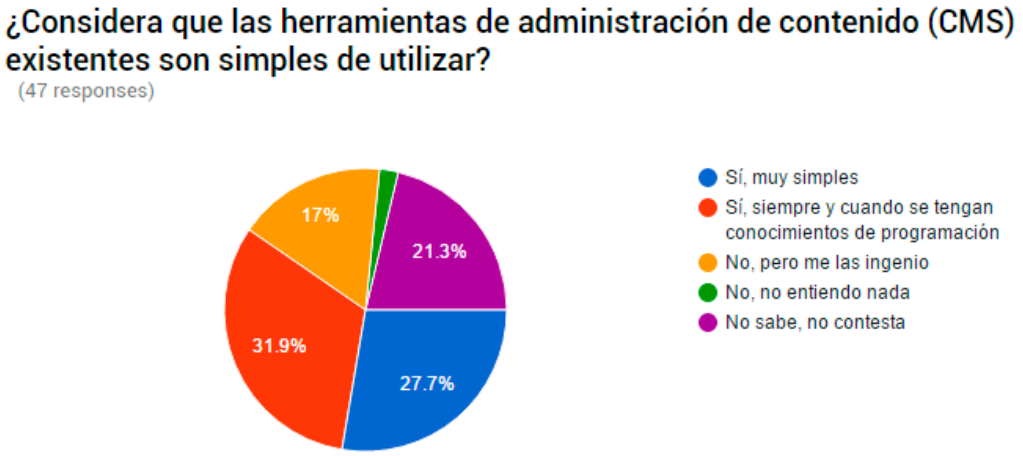
También es importante notar que casi el 100% de los encuestados están familiarizados con tecnologías web, como HTML, CSS y JavaScript.




Mediante esta pregunta ratificamos los datos evaluados acerca de la competencia, con WordPress a la cabeza.

Esto nos permite suponer que la mayor parte de los encuestados están familiarizados con el concepto de CMS y conocen la finalidad y funcionalidades que provee este tipo de software.

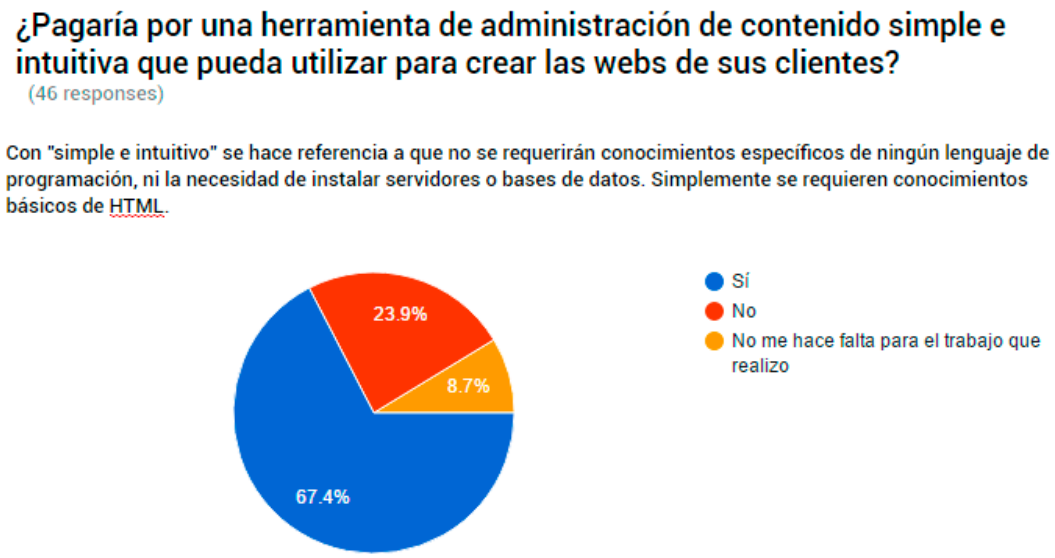
Vemos que dentro de los encuestados hay pocos que conocen acerca de Joomla o Drupal ya que los mismos son utilizados por programadores para desarrollos más específicos que requieren mayor nivel de personalización.




Una cuarta parte de la población encuestada no conoce lo que es un CMS, o bien nunca ha tenido que utilizar alguno. Probablemente sea una opción marcada por programadores que se dedican al desarrollo a medida, o por diseñadores gráficos que no desconocen estas alternativas. Es probable que en este último caso los diseñadores no se dediquen al diseño de webs, que las webs que han tenido que diseñar no poseen contenido dinámico o administrable, o bien han derivado esta tarea con profesionales de las áreas de sistemas.

Más de la cuarta parte manifiesta que este tipo de herramientas requieren conocimientos de programación. Probablemente representa a los diseñadores gráficos que se ven intimidados por el uso de estas tecnologías.

Por último, el resto de la población indica que sabe cómo integrar y configurar un CMS, ya sea porque posee los conocimientos técnicos necesarios o bien porque consideran por experiencia que implementar estas soluciones se puede aprender.




A través de esta última pregunta obtenemos feedback acerca de la posibilidad de adopción de una solución de CMS que le evite al usuario enfrentarse a las complicaciones que presenta la instalación de servidores y conocimientos de lenguajes de programación.

La porción de la población que no le interesa la propuesta es probable que no la necesite para su trabajo o bien considera que pagar no es apropiado siendo que en el mercado existen muchas opciones gratuitas que se pueden utilizar.

Este es un punto a tener en cuenta: la posibilidad de armar un modelo comercial que contemple el uso de licencias gratuitas, por ejemplo accediendo a una versión básica con funcionalidades limitadas, o bien pudiendo descargar el código fuente y publicar la solución en un servidor propio del usuario (como lo hacen los principales CMS de código abierto)

## 15.2. Relevamiento estructural

Dado el perfil del usuario se asume que los mismos poseen computadora con conexión a internet que les permite desenvolverse en su trabajo diario.

El sistema será accesible vía web y no requiere grandes capacidades a nivel de hardware en la pc del usuario. Tampoco se requiere la instalación de software adicional al utilizado para la construcción de cualquier sitio web: un editor de texto y un navegador web.

## 15.3. Relevamiento funcional

A la hora de implementar una solución CMS de cualquier tipo existen por lo menos los siguientes roles básicos:

- **Desarrollador:** se encarga del desarrollo de la web y la integración de contenidos desde el CMS. Configura e implementa el sistema CMS en un servidor web. Crea las estructuras básicas e instala los componentes o extensiones necesarios para poder resolver los requerimientos de su cliente.

- **Administrador:** se encarga de administrar los contenidos en el CMS que luego serán mostrados en la web. Puede dar de alta, baja o editar un contenido. Puede dar de alta categorías, jerarquizar y ordenar la información. Puede dar permisos de acceso a otros usuarios que lo ayuden en esta tarea. La tarea que desarrolla es por lo general operativa y no requiere conocimientos técnicos de sistemas.

- **Usuario web:** no tiene participación en el uso del sistema CMS, pero se lo incluye a modo de referencia ya que es quien inicializa el proceso para obtener el contenido y visualizarlo en su navegador web.

## 16. Procesos de negocios

Antes de continuar con los procesos cabe aclarar que en los mismos se involucran 2 aplicaciones fundamentales que es necesario conocer y relacionar a los mismos para una mejor comprensión:

- **Aplicación Backend:** permite la gestión de los contenidos almacenando la información en una base de datos. Sólo puede ser accedida por el usuario administrador (y eventualmente también por el desarrollador).

- **Aplicación Frontend:** permite visualizar los contenidos cargados desde la aplicación Backend. Los contenidos se muestran embebidos en una plantilla predefinida configurada por el desarrollador o por el administrador del sitio. A esta aplicación acceden los usuarios del sitio web. No pueden editar el contenido.

### 16.1. Proceso 1 – Instalación del CMS

**Roles:** Desarrollador

**Aplicación:** Backend y Frontend

**Pasos:**

1. El usuario descarga el código fuente del CMS de la web del proveedor;
2. El usuario accede por FTP al servidor en donde estará publicado el sitio web (previamente contratado);
3. El usuario copia el contenido en el servidor;
4. El usuario accede al panel de configuración del CMS;

5. El usuario configura la conexión a la base de datos y ejecuta los comandos de creación provistos por el CMS;
6. El usuario personaliza los ítems de menú, roles, usuarios y configuración básica de template y temas según requerimiento del cliente.
7. El usuario accede con la URL del Frontend para corroborar que el sitio funciona correctamente;
8. Fin.

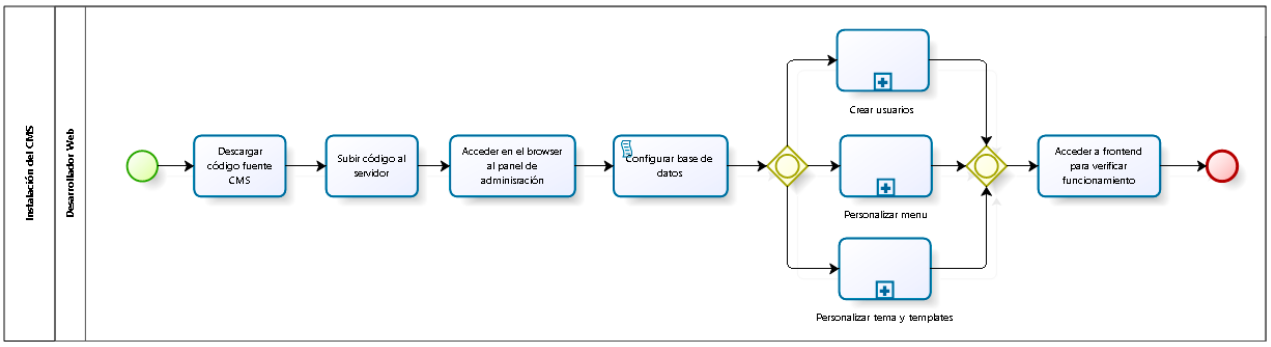



*Figura 2: Proceso de instalación de un software CMS tradicional*

### 16.2. Proceso 2 - Login

**Roles:** Administrador

**Aplicaciones:** Backend

**Pasos:**

1. El usuario ingresa a la plataforma por medio de un navegador web;
2. Si no está logueado el sistema muestra un formulario para ingresar usuario y password, sino prosigue con paso 3;
    a. El usuario envía los datos del formulario
    b. El sistema valida los datos. Si son correctos prosigue con paso 3, de lo contrario se informa el error y permanece en paso 2;
3. El sistema dirige al usuario a la pantalla de inicio y muestra las opciones habilitadas para su rol;
4. Fin.

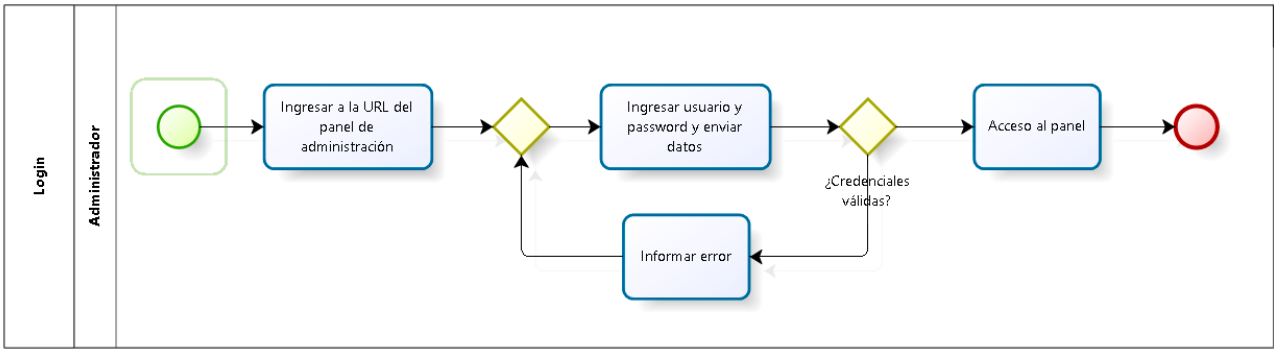



*Figura 3: Login de usuario en CMS*

### 16.3. Proceso 3 - Gestión de contenido

**Roles:** Administrador

**Aplicaciones:** Backend

**Pasos:**

1. El usuario accede al sistema;
2. El usuario accede al listado de artículos;
3. El usuario puede crear un nuevo artículo o editar o eliminar uno ya existente;
    a. Si elimina el artículo se le pide confirmación y se elimina;
    b. Si crea o edita se muestra un formulario predefinido en donde puede cargar o actualizar título, contenido, imágenes, links, etiquetas, categoría y otras opciones de visualización. Luego guarda los cambios;
4. El sistema actualiza el listado de artículos de acuerdo a si la acción realizada fue de tipo alta, baja o actualización;
5. Fin;

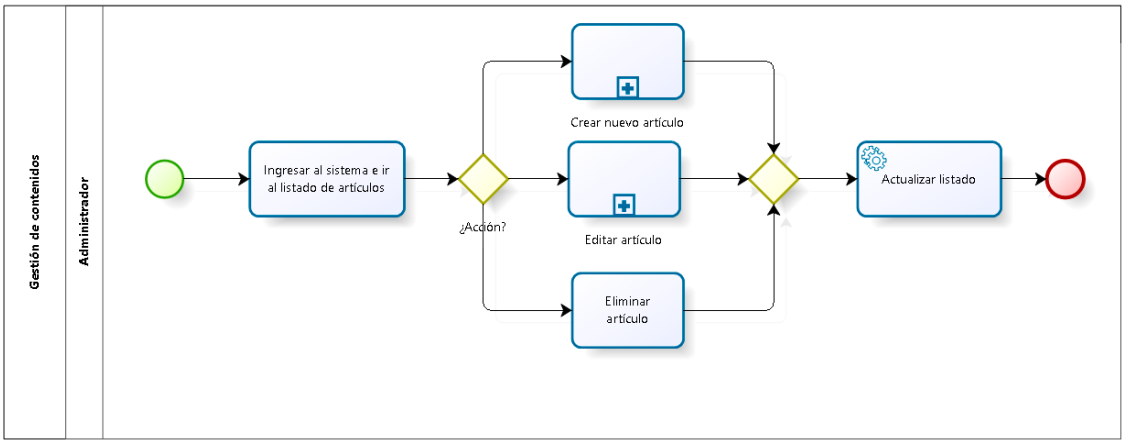



*Figura 4: Proceso de gestión de contenido en un CMS tradicional*

### 16.4. Proceso 4 - Gestión de categorías

**Roles:** Administrador

**Aplicación:** Backend

**Pasos:**

1. El usuario accede al sistema;
2. El usuario accede al listado de categorías;
3. El usuario puede crear una nueva categoría o editar o eliminar una ya existente;
    a. Si elimina la categoría se le pide confirmación y se elimina. Los artículos vinculados a dicha categoría quedan sin categorías;
    b. Si crea o edita se muestra un formulario en donde puede cargar o actualizar el nombre de la categoría y un alias a mostrar en la url. Además puede seleccionar de una lista uno o más artículos a los cuales se vinculará la categoría. Luego guarda los cambios;
4. El sistema actualiza el listado de categorías de acuerdo a si la acción realizada fue de alta, baja o actualización;
5. Fin;

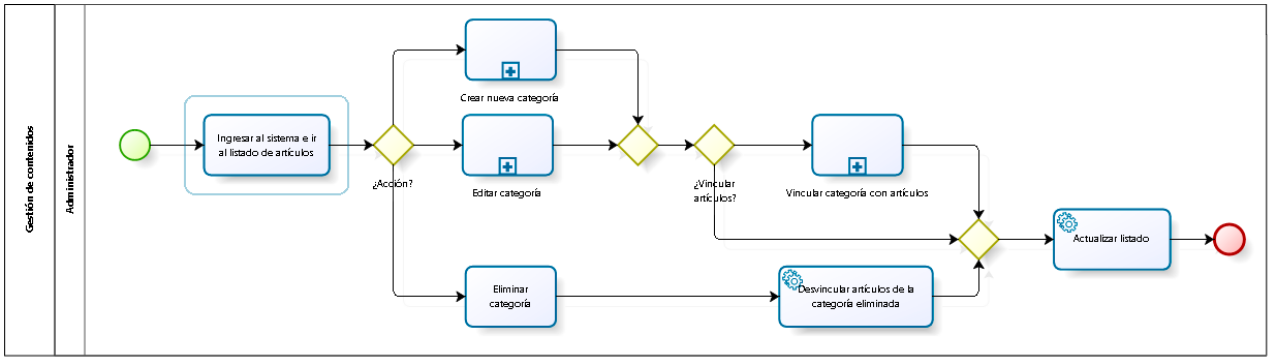



*Figura 5: Proceso de gestión de categorías en un CMS tradicional*

## 16.5. Proceso 5 - Personalización del diseño

**Roles:** Administrador y/o desarrollador

**Aplicación:** Backend

**Pasos:**

1. El usuario se loguea al panel de administración;
2. El usuario ingresa a la sección de templates:
    a. El usuario puede buscar y cambiar los templates para las distintas secciones eligiendo de entre una lista provista por el CMS;
    b. El usuario puede editar el código fuente de los templates para obtener una mayor personalización;
3. El usuario ingresa a la sección de temas:
    a. El usuario puede buscar y cambiar el tema actual eligiendo de entre una lista provista por el CMS y que estén disponibles para el template seleccionado;
    b. El usuario puede editar la configuración general del tema definiendo estilos de letra (tamaño, fuente, etc), colores de los textos y fondo, imágenes, logo, entre otros.
4. El usuario ingresa a la sección de administración de menú:
    a. El usuario puede agregar, editar, eliminar o reordenar los enlaces del menú;
5. El usuario accede al lisado de complementos o extensiones:
    a. El usuario puede remover o configurar extensiones existentes;
    b. El usuario puede buscar nuevas extensiones para instalar;
6. Fin.

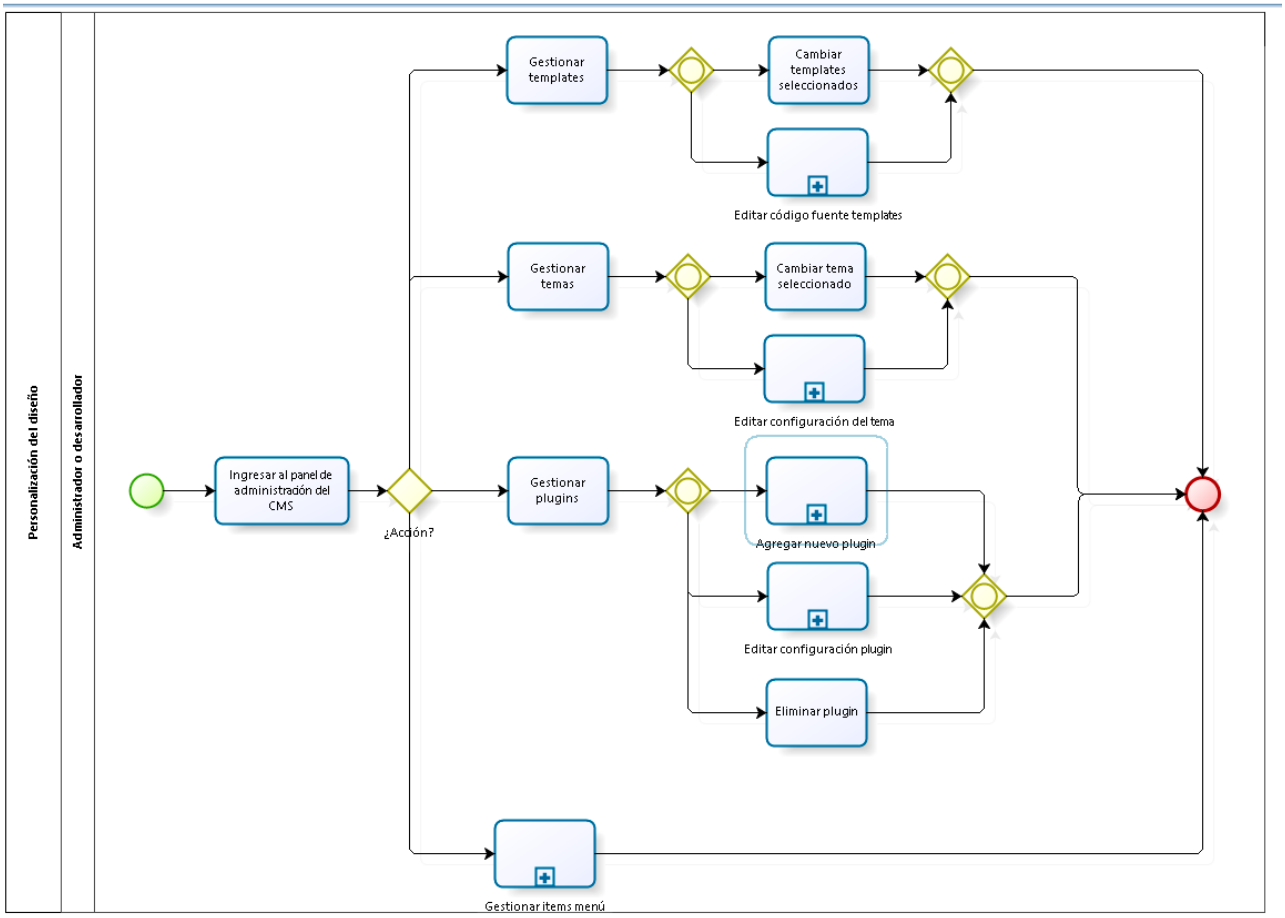



*Figura 4: Proceso de personalización de diseño en un CMS tradicional*

## 16.6. Proceso 6 - Visualización del contenido

**Roles:** Usuario

**Aplicación:** Frontend

**Pasos:**

1. El usuario ingresa a la web mediante una URL;
2. El sistema identifica el recurso requerido por el usuario y obtiene de la base de datos un lisado de artículos y/o el detalle de un artículo;
3. El sistema embebe el contenido obtenido en un template y devuelve al navegador del usuario el contenido HTML generado;
4. El usuario visualiza el contenido solicitado en su navegador;
5. Fin.

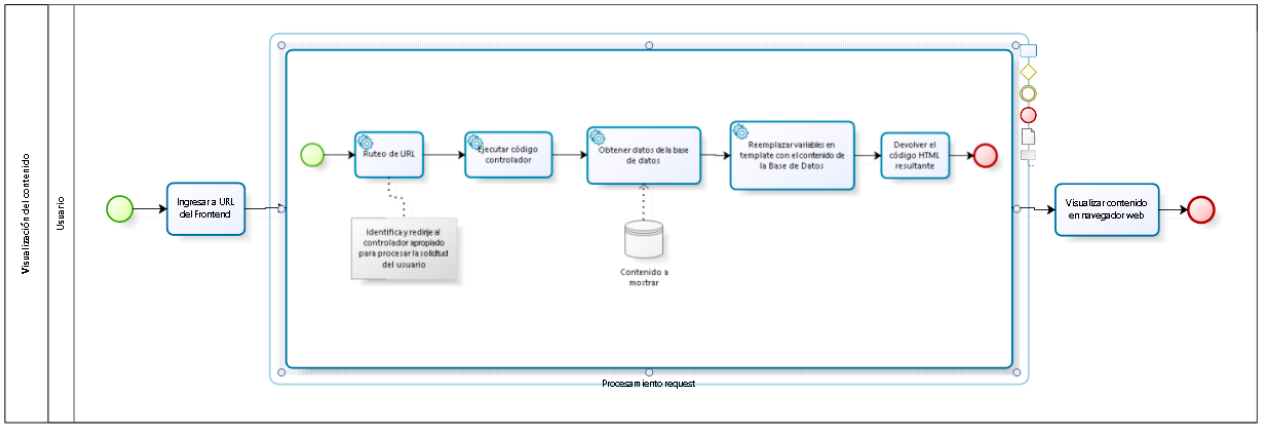



*Figura 5: Proceso de visualización del contenido en un CMS tradicional*

# 17.Diagnóstico

En líneas generales los procesos convencionales funcionan correctamente para el modelo de solución clásico propuesto por la mayoría de los CMS.

No obstante estas soluciones involucran la participación de un desarrollador más o menos experimentado, cuyas habilidades requeridas variarán en función del nivel de personalización deseado para el sitio web a construir.

Entonces, lo que se propone identificar en los procesos existentes son oportunidades de mejora que permitan facilitar y simplificar el uso del CMS, e innovando principalmente en aquellos aspectos que puedan verse beneficiados mediante la implementación de nuevas tecnologías.

## 17.1. Proceso 1 - Instalación del CMS

La complejidad técnica vinculada a la instalación y subida del sitio a un servidor hoy en día es abordada por algunas de las soluciones de CMS más populares como WordPress o Joomla los cuales ofrecen la posibilidad de gestionar este proceso directamente en la nube brindando hosting en sus servidores, evitando así el proceso alta y configuración del sistema y base de datos. No obstante, esta solución no posibilita tener un entorno de desarrollo local, por lo que a grandes rasgos es recomendada sólo para usuarios convencionales con nula experiencia en desarrollo web. Tanto el código del Backend como del Frontend se encuentra completamente en la nube.

Además, como ya se analizó antes, estas soluciones están totalmente integradas a la aplicación Frontend, lo cual puede resultar en una desventaja al no poder integrar los contenidos en otra tecnología o sistema.

**Oportunidad de mejora:**

Se considera que la aplicación Frontend debería ser independiente de la aplicación Backend, pudiendo la primera ser desarrollada en cualquier tecnología o plataforma e integrando los datos provistos por la segunda.

Además se considera que el código del Frontend debería estar 100% en control del desarrollador, facilitando su customización, versionado y mantenimiento.

## 17.2. Proceso 2 - Login

No presenta inconvenientes en líneas generales.

## 17.3. Proceso 3 y 4 - Gestión de contenido y gestión de categorías

Es sabido que la mayoría de los CMS nacieron inicialmente como herramientas para crear y dar soporte a Blogs (sitios web con entradas o artículos ordenados cronológicamente a modo de bitácora).

Con el paso del tiempo los paradigmas y requisitos de los usuarios fueron cambiando y estas herramientas de blogging comenzaron a dar soporte a la creación de páginas web, menús de navegación, etc.

No obstante esto, la estructuras de datos que permiten gestionar por lo general están limitadas a los elementos de una noticia, como son: título, subtítulo, descripción, contenido, autor, fecha, imágenes, videos, categorías (taxonomías), etiquetas, entre otros.

Si como usuario deseo tener por ejemplo un sitio web de autos entonces las herramientas de las que dispondré en principio serán limitadas, aunque para los que tengan más conocimiento y experiencia es factible utilizar extensiones de terceros que permiten generar estructuras con contenido más personalizable.

**Oportunidad de mejora:**

Se considera que un sistema de gestión de contenido debería permitir al usuario gestionar de manera dinámica los atributos y tipos de campo (textuales, numéricos, multimedia, etc.) que caracterizarán a cada entidad que necesiten administrar.

## 17.4. Proceso 5 - Personalización del diseño

Los CMS más populares ofrecen una gran cantidad de opciones de personalización en cuanto a temas, templates, colores, fuentes, ubicación del contenido, etc.

Además, mediante el uso de extensiones de terceros las posibilidades de personalización, tanto estéticas como también a nivel de funcionalidades, se incrementa mucho más.

El problema es que más allá del gran abanico de opciones de configuración existentes las mismas resultan insuficientes a la hora de implementar un diseño personalizado que respete al 100% lo planteado por un diseñador gráfico.

Siempre en estos casos se requiere editar los templates HTML, las hojas de estilos CSS y posiblemente algún archivo JavaScript. Este proceso resulta complejo y difícil de mantener por los siguientes motivos:

- En la mayoría de los casos los templates (capa de vista) tienen embebidos lógica de negocio;
- En otros casos (sobre todo cuando se usan extensiones) los templates se encuentran guardados en campos de una base de datos;
- Por lo general, con la idea de dar soporte a una variada cantidad de funcionalidades, los templates sobre los que se trabajan contienen embebidas muchas más funcionalidades de las que el usuario realmente hará uso.

**Oportunidad de mejora:**

Se profundiza en la idea analizada en el proceso 1 donde se sugirió una separación clara entre Frontend y Backend. El código de la web debería ser generado por el usuario en la medida que satisfaga sus necesidades, integrando desde el Backend sólo los datos de las entidades administradas.

## 17.5. Proceso 6 - Visualización del contenido

Cuando un usuario navega hacia una de las páginas administradas por el CMS es el servidor quien procesa este pedido combinando un template con datos dinámicos. Luego el código HTML resultante es devuelto al navegador web quien lo procesa y renderiza para que sea visible y manipulable por el usuario.

Esto genera una gran sobrecarga en el servidor quien tiene múltiples responsabilidades, por ejemplo:

- Conectarse con la base de datos para obtener los datos solicitados;

- Dichos datos, por lo general, son devueltos en formato de filas y columnas por lo que se deben procesar para poder reconstruir las estructuras originales antes de poder ser manipuladas;
- Luego de obtener las entidades debe procesar el template y reemplazar las variables por los datos obtenidos;
- Por último debe devolver el código generado, junto a cualquier otro recurso web vinculado, como hojas de estilos, imágenes o scripts.

**Oportunidad de mejora:**

Se considera que las siguientes tareas podrían optimizarse:

- Por un lado evitar la necesidad de tener que reconstruir constantemente las entidades a partir de los datos almacenado en las tablas de la base de datos;
- Delegar la responsabilidad de renderizar los templates al cliente web.

# 18. Propuestas de solución

## 18.1. Propuesta de solución general

Tomando como base la problemática relevada y el diagnóstico realizado se propone analizar, desarrollar e implementar un sistema CMS que adopte tecnologías y metodologías modernas que permitan una mejor integración, flexibilidad y experiencia general de uso.

Se tomará como punto de partida cada uno de los procesos diagnosticados para proponer una solución a cada una de las oportunidades de mejora detectadas.

El producto CMS a desarrollar le permitirá a uno o más usuarios administrar las entidades y contenidos requeridos por su sitio web. Esta gestión se realizará de manera online para que no requiera la instalación o configuración de un ambiente de desarrollo en su propia PC, algo que se considera de cierta complejidad relativa para muchos usuarios que no son desarrolladores.

Cada usuario podrá crear y administrar el contenido para uno o más sitios web, los cuales llevarán el nombre de Aplicaciones en nuestro sistema. Cada Aplicación tendrá asignada su propia base de datos y en ella se guardarán las entidades y datos correspondientes a la misma.

Luego, para integrar el contenido de cada una de las Aplicaciones los usuarios podrán utilizar una Librería Cliente que conectará su sitio web con el sistema CMS y le permitirá obtener los datos almacenados según corresponda. Esta librería estará desarrollada íntegramente en JavaScript para permitir su uso en cualquier página web, sin importar la plataforma o lenguaje de programación de servidor utilizado.

A continuación se describe más en detalle la propuesta de mejora sobre cada uno de los procesos relevados.

### 18.1.1. Proceso 1 - Integración del CMS

Notar que se habla de integración en lugar de instalación. Lo que se propone es que el usuario pueda crear una cuenta de Aplicación en el CMS. Esta cuenta le habilitará un panel de administración para que pueda gestionar todo su contenido en línea. La base de datos y conexión será creada y configurada automáticamente para evitar esta complejidad técnica.

Luego, el usuario podrá integrar el contenido administrado en su sitio web utilizando una Librería Cliente provista como parte de la solución. Esta librería estará desarrollada en JavaScript, lenguaje interpretado por cualquier navegador web, lo que permitirá su uso en cualquier sitio web.

El usuario definirá la estructura HTML y estilos en su sitio web (como lo hace normalmente) e identificará las secciones en las cuales se desea incorporar los datos desde el CMS mediante el simple uso de etiquetas y atributos HTML (los cuales serán predefinidos y reconocidos por la librería a desarrollar). Dicha estructura se utilizará como template para mostrar el contenido dinámico.

**Pasos:**

1. El usuario desarrollador crea una cuenta que le permite acceder al panel de administración de su Aplicación en el CMS;
2. El usuario se loguea al CMS (según se describe en detalle en el proceso 2);
3. El usuario gestiona el contenido (según se describe en detalle en proceso 3);
4. El usuario incorpora el script de la librería cliente en su sitio web;
5. El usuario configura el identificador de la aplicación que le permitirá establecer la conexión con el servicio CMS;
6. El usuario incorpora en su sitio las etiquetas y/o atributos que le permitirán integrar sus contenidos;
7. Fin.

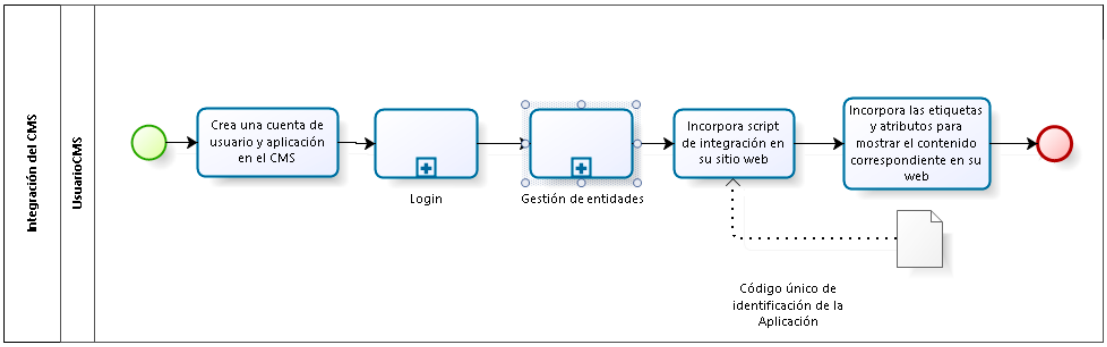



*Figura 6: Proceso de integración del CMS en la web del usuario*

### 18.1.2. Proceso 2 - Login

Se requiere implementar acceso restringido al panel de administración del CMS desde donde el usuario gestionará el contenido.

**Pasos:**

1. El usuario accede a la URL del panel de administración del CMS;
2. El usuario ingresa su usuario y contraseña;
3. El sistema valida las credenciales y permite el acceso;
4. Fin.

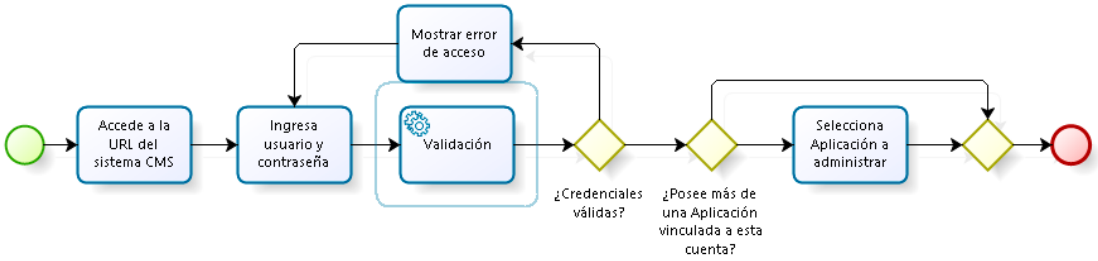



*Figura 7: Proceso de login en el panel de administración del CMS*

### 18.1.3. Proceso 3 - Gestión de contenido

Como ya se anticipó antes, el contenido será administrado desde una plataforma en la nube.

Atento a resolver la problemática de gestionar estructuras dinámicas es que se optará por el uso de un sistema de base de datos de tipo NoSQL.

Este proceso se descompondrá en 2 subprocesos que se definen a continuación:

#### 18.1.3.1. Subproceso 3A - Gestión de la estructura del contenido (entidades):

Dado que las estructuras a gestionar serán dinámicas el usuario deberá indicar al sistema los atributos que componen a dichas entidades.

Por ejemplo, se podrá crear una colección de entidades de nombre “Noticia” en la que se definan los siguientes atributos:

- Título, de tipo texto;
- Imagen, de tipo imagen;
- Descripción: de tipo texto enriquecido.

**Pasos:**

1. El usuario ingresa al listado de entidades y selecciona una ya existente o bien puede crear una nueva ingresando un identificador para la misma;
2. El usuario hace clic en un botón que le permite cargar un nuevo atributo;
3. El usuario ingresa un identificador para el atributo y selecciona un tipo de dato de entre los proporcionados por el sistema;
4. El usuario puede seguir agregando nuevos atributos repitiendo los pasos 2 y 3, o finalizar para generar la estructura deseada;
5. Fin.

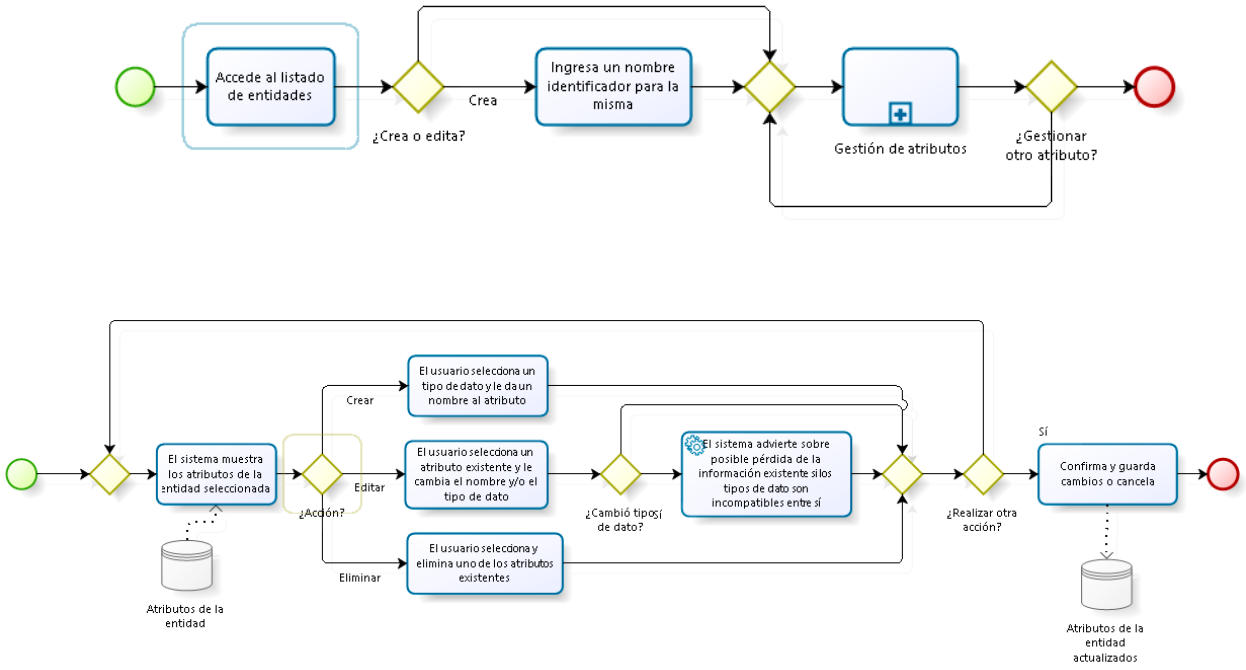



*Figura 8: Subproceso para la creación o actualización de entidades*

### 18.1.3.2. Subproceso 3B - Gestión del contenido propiamente dicho:

Valiéndose de la estructura definida en el subproceso anterior el sistema generará automáticamente un formulario para que el usuario pueda dar de alta el contenido indicado.

**Pasos:**

1. El usuario ingresa al listado de entidades y selecciona una;
2. El usuario hace clic en un botón que le permite agregar nuevo contenido para dicha entidad;

3. El sistema genera un formulario de manera dinámica por cada atributo configurado para la entidad. El tipo de campo a mostrar variará de acuerdo al tipo de atributo;
4. El usuario completa el formulario y hace clic en un botón para guardar;
5. El sistema crea una nueva entidad a partir de los datos recibidos del formulario. Luego inserta esa entidad en la colección correspondiente en la base de datos.
6. El sistema retorna al listado de contenidos de la entidad, en donde se incluye el nuevo contenido cargado;
7. Fin.

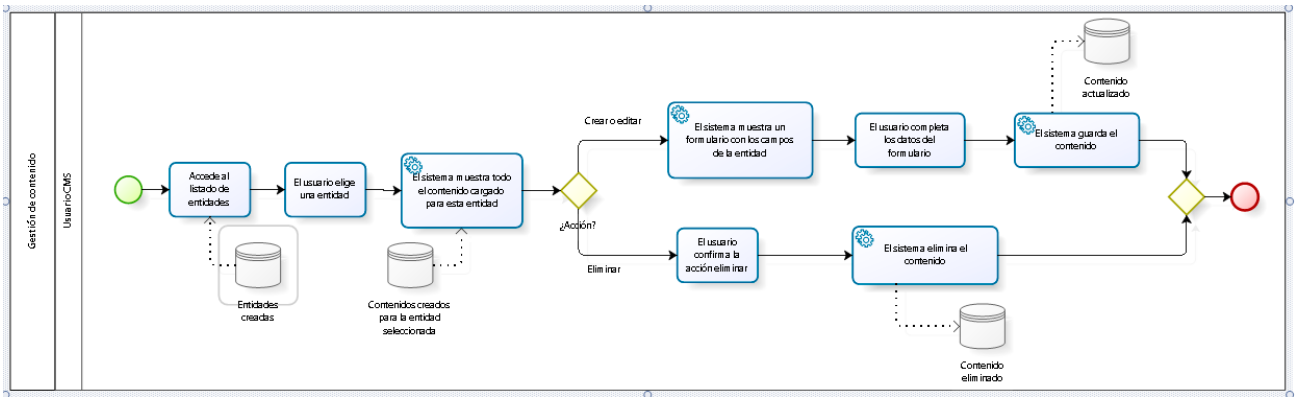



*Figura 9: Subproceso de creación de contenido a partir de una entidad*

### 18.1.4. Proceso 4 – Visualización del contenido

**Pasos:**

1. Un Usuario Web accede con su navegador a la web que integra los contenidos del CMS;
2. El servidor del cliente devolverá el código HTML de su página, en el que se incluye la librería que se integra con la Aplicación en el sistema CMS;
3. La librería identifica las etiquetas y/o atributos que le indican que debe realizar una petición al servidor para obtener datos de una entidad o colección de entidades;
4. La librería procesa e imprime en pantalla el contenido HTML generado a partir del template y los datos obtenidos;
5. El usuario visualiza el contenido en pantalla.
6. Fin.

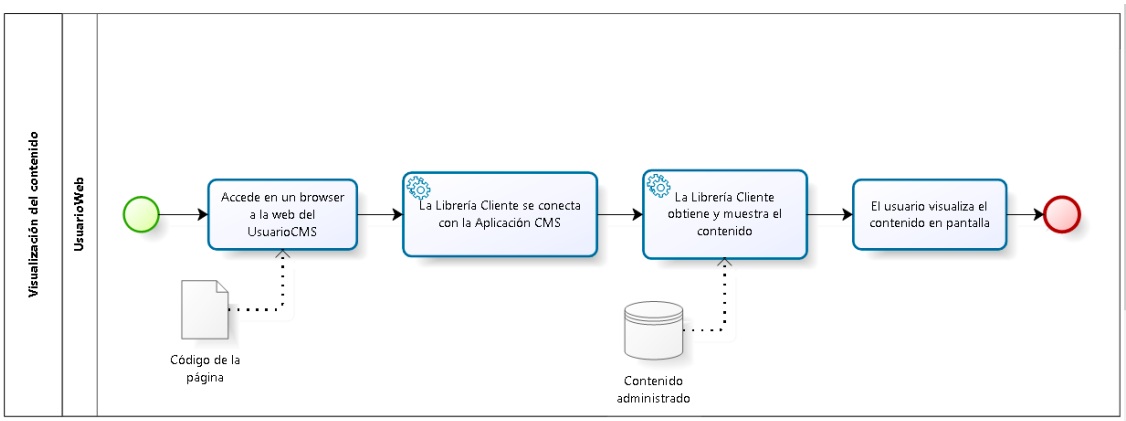



*Figura 10: Proceso de visualización de contenido en la web*

## 19. Requerimientos del sistema

La solución estará dividida en 3 subsistemas con el objetivo de facilitar la estructuración del código, la estimación y el seguimiento de las tareas:

- **Aplicación Backend:** permitirá realizar la gestión de todas aquellas entidades y procesos que brindarán el soporte necesario para que los usuarios del CMS puedan gestionar luego el contenido de sus aplicaciones.

  Podrán acceder a ella los usuarios con el rol ADMINISTRADOR_SISTEMA.

- **Aplicación CMS:** a esta aplicación accederán los usuarios del CMS para administrar el contenido de su aplicación web.

  Podrán acceder a ella los usuarios con el rol USUARIO_CMS.

- **Librería Cliente:** involucra todo el desarrollo necesario para que un usuario del CMS pueda integrar el contenido gestionado en su propia web.

### 19.1. Requerimientos funcionales

A continuación se listarán los requerimientos funcionales relevados por subsitema:

**Aplicación Backend:**

RF_1 El sistema debe permitir inicio de sesión con usuario y contraseña para el rol ADMINISTRADOR_SISTEMA.

RF_2 El sistema debe permitir gestionar información de:

- Usuarios;
- Aplicaciones.

**Aplicación CMS:**

RF_3 El sistema debe permitir inicio de sesión con usuario y contraseña para el rol USUARIO_CMS.

RF_4 El sistema debe permitir el alta de usuarios y de aplicaciones por usuario.

RF_5 Al dar de alta una nueva aplicación se generarán automáticamente las credenciales para la integración con la librería cliente

RF_6 El sistema debe permitir al usuario gestionar información de:

- Entidades;
- Contenido por entidad;
- Datos de la aplicación;
- Perfil del usuario;

RF_7 Cada entidad administrada podrá tener asociados un número N de atributos, pudiendo cada campo almacenar información de uno de los siguientes tipos: texto simple, texto enriquecido, imagen, número, opciones excluyentes (radio-button), opciones simples (checkbox) y fecha.

**Librería Cliente:**

RF_8 Establecer conexión / autenticación con la Aplicación CMS correspondiente;

RF_9 Obtener los datos de una colección de entidades o de una entidad en particular

RF_10 Mostrar el contenido obtenido de la o las entidades utilizando como template un fragmento de la web del cliente.

### 19.2. Reglas de negocio

- Sólo tendrán acceso a la Aplicación Backend los usuarios con el rol ADMINISTRADOR_SISTEMA;
- Sólo tendrán acceso a la Aplicación CMS los usuarios con el rol USUARIO_CMS;
- Un usuario de rol USUARIO_CMS sólo podrá gestionar entidades y contenidos pertenecientes a su aplicación;

- Un usuario de rol USUARIO_CMS sólo podrán gestionar el contenido de su aplicación siempre y cuando la misma esté activa.
- La librería cliente podrá obtener el contenido almacenado en la Aplicación CMS por más que la aplicación esté inactiva, a menos que se deniegue explícitamente este permiso.

## 19.3. Requerimientos no funcionales

RFN_1 Incorporar ayuda en línea para los componentes o funcionalidades que se consideren necesarios.

RFN_2 El contenido administrado en el CMS deberá poder ser integrado en cualquier página web, sin importar el lenguaje de programación de servidor utilizado.

RFN_3 El peso de descarga de la librería cliente deberá ser inferior a los 300Kb para no repercutir negativamente en la performance de carga del resto del sitio.

## 19.4. Requerimientos candidatos

**Aplicación Backend:**

RC_1 Sección de estadísticas de cantidad de usuarios y visitas por cada una de las aplicaciones administradas por el sistema.

**Aplicación CMS:**

RC_2 Soporte de nuevos tipos de datos / campos para la administración de entidades. Ej: video, galería, etiquetas;

RC_3 Permitir la gestión de contenido en múltiples idiomas;

RC_4 Permitir la incorporación de nuevas funcionalidades mediante plugins y extensiones desarrolladas por terceros.

RC_5 Sección de estadísticas de cantidad de usuarios y visitas por cada una de las aplicaciones del usuario.

RC_6 Definir distintos roles y permisos de usuarios por aplicación.

**Librería Cliente:**

RC_7 Permitir la edición del contenido directamente desde la web del cliente.

# 20. Desarrollo del Producto/servicio

## 20.1. Historias de Usuario

A continuación se listan las historias de usuario agrupadas por subsistema y usuario a cual va destinada la funcionalidad.

Dichas historias de usuario representan la totalidad de funcionalidades contempladas para el desarrollo del producto.

### 20.1.1. Historia – BACKEND_001

**Nombre corto:** Login administrador.

**Descripción:** Como administrador del sistema deseo poder ingresar con mi usuario y contraseña para luego poder gestionar las aplicaciones registradas y usuarios del sistema.

**Criterios de aceptación:**

- Si ingreso un usuario o contraseña incorrecto se debe notificar el error;
- Si ingreso un usuario y contraseña válidos debo poder acceder al sistema y visualizar el menú correspondiente para gestionar aplicaciones y usuarios.

### 20.1.2. Historia – BACKEND_002

**Nombre corto:** Administración de cuentas de usuario.

**Descripción:** Como administrador del sistema deseo poder gestionar la información de todos los usuarios, así como también crear nuevos.

**Criterios de aceptación:**

- Al ingresar al listado de usuarios debo poder visualizar en una grilla la información de los mismos: nombre, usuario, estado (activo o inactivo).
- Al seleccionar un usuario debo poder editar y actualizar los datos de los mismos: usuario, contraseña, nombre, email, rol, estado (activo o inactivo).
- Si al editar los datos del usuario algunos campos obligatorios están incompletos o si cargo mal algún tipo de dato debo poder ver el error en pantalla para poder corregirlo.
- El nombre de usuario debe ser único.
- La contraseña del usuario no se debe visualizar. Sin embargo se podrá actualizar ingresando una nueva.
- La contraseña se debe almacenar encriptada en la base de datos.
- Luego de dar de alta un nuevo usuario debe aparecer en la lista de usuarios con el rol correspondiente.

### 20.1.3. Historia - BACKEND_003

**Nombre corto:** Administración de aplicaciones.

**Descripción:** Como administrador del sistema deseo poder gestionar las aplicaciones generadas por los usuarios del CMS.

**Criterios de aceptación:**

- Al ingresar al listado de aplicaciones debo poder visualizar en una grilla la información de las mismas: nombre, identificador, usuario creador, fecha de alta, estado (activa o inactiva).
- Al seleccionar una aplicación debo poder visualizar los datos completos de la misma junto con los datos del usuario creador.
- Al seleccionar el usuario de la aplicación me abrirá la pantalla de edición de los datos del mismo.

### 20.1.4. Historia - CMS_001

**Nombre corto:** Login usuario CMS

**Descripción:** Como usuario del CMS deseo poder ingresar con mi usuario y contraseña para luego poder gestionar mis aplicaciones registradas.

**Criterios de aceptación:**

- Si ingreso un usuario o contraseña incorrectos se debe notificar el error;
- Si ingreso un usuario y contraseña válidos debo poder acceder al sistema y visualizar el listado de las aplicaciones que tengo asociadas, mostrando los siguientes campos: nombre, identificador, fecha de alta, estado (activa o inactiva).

### 20.1.5. Historia - CMS_002

**Nombre corto:** Ingreso a panel de aplicación.

**Descripción:** Como usuario del CMS deseo poder gestionar los datos de mis aplicaciones, así como también las entidades y contenidos.

**Criterios de aceptación**

- Al ingresar a la cuenta el sistema mostrará un listado de aplicaciones creadas por el usuario.
- El sistema debe mostrar un botón para poder crear una nueva aplicación.

- Al seleccionar una aplicación en estado inactivo el sistema deberá indicar al usuario que la aplicación fue dada de baja y debe ponerse en contacto con los administradores del sistema;
- Al seleccionar una aplicación en estado activo el sistema mostrará un menú para que el usuario pueda gestionar los datos de la aplicación, entidades y contenidos vinculados.

### 20.1.6. Historia - CMS_003

**Nombre corto:** Crear aplicación.

**Descripción:** Como usuario del CMS deseo poder dar de alta una nueva aplicación.

**Criterios de aceptación:**

- El nombre de la aplicación debe tener menos de 20 caracteres, caso contrario mostrará un error.
- Al crear la aplicación el sistema generará y asociará de forma automática un código numérico de identificación y una clave alfanumérica de tipo GUID de 36 caracteres.
- El identificador de la aplicación debe ser único e irrepetible. La clave no necesita ser única.
- Al crear la aplicación el sistema vinculará la misma al usuario logueado.

### 20.1.7. Historia - CMS_004

**Nombre corto:** Administrar datos de la aplicación.

**Descripción:** Como usuario del CMS deseo poder visualizar y gestionar los datos de la aplicación seleccionada.

**Criterios de aceptación:**

- Al ingresar a ver los datos de la aplicación debo poder visualizar los siguientes datos: nombre, identificador, clave, estado (activa o inactiva).
- Al ingresar a la edición de los datos de la aplicación sólo podrá actualizar en forma directa el campo nombre;
- Al ingresar a la edición de los datos de la aplicación el usuario podrá actualizar la clave alfanumérica mediante un botón que genere de forma automática una nueva clave.
- Al regenerar la clave alfanumérica se deberá advertir al usuario antes de guardar los cambios de que al hacer esto deberá actualizar la configuración de

conexión en las webs en donde se estén integrando los contenidos de dicha aplicación.

### 20.1.8. Historia - CMS_005

**Nombre corto:** Administrar entidades de la aplicación.

**Descripción:** Como usuario del CMS deseo poder visualizar y gestionar las entidades para cada aplicación.

**Criterios de aceptación:**

- Al ingresar a ver las entidades creadas de la aplicación debo poder visualizar las mismas listadas por nombre.
- Al crear una nueva entidad o editar una existente deberé poder configurar un nombre y los atributos que la componen.
- El nombre de la entidad debe ser único para esta aplicación.
- El usuario podrá vincular uno o más atributos a la entidad.
- Por cada atributo agregado el usuario deberá indicar un nombre y elegir entre uno de los siguientes tipos de dato preestablecidos: texto simple, texto enriquecido, imagen, número, opciones excluyentes (radio-button), opciones simples (checkbox) y fecha.
- Por cada atributo se podrá indicar si el mismo es requerido o no.
- El usuario debe poder cambiar el orden de los atributos.
- Al editar un campo no se podrá cambiar su tipo de dato.
- Se debe pedir la confirmación del usuario al eliminar un atributo.
- En caso de eliminar un atributo esto tendrá efecto sólo para la creación de nuevos contenidos, pero no debe afectar a los contenidos ya existentes.

### 20.1.9. Historia - CMS_006

**Nombre corto:** Administrar contenido por entidad.

**Descripción:** Como usuario del CMS deseo poder visualizar y gestionar contenido por cada entidad.

**Criterios de aceptación:**

- Se debe contar con un menú de acceso a los contenidos por cada entidad existente;
- Al ingresar a ver los contenidos de una entidad se podrá visualizar un listado de los contenidos mostrando los campos que hayan sido marcados para mostrar en la grilla (en la configuración del atributo de la entidad).

- Se podrá seleccionar un contenido para editarlo o eliminar el mismo luego de aceptar la confirmación del sistema.
- Al crear o editar un contenido el sistema mostrará un formulario de carga con los atributos asignados a la entidad a la que pertenece el contenido.
- Los atributos dentro del formulario serán presentados en el orden configurado en la entidad.
- Por cada atributo se mostrará en el formulario un componente que me permita cargar o asignar un valor. Este componente dependerá del tipo de dato vinculado al atributo en la entidad.
- El sistema deberá validar que todos los datos marcados como requeridos en la entidad hayan sido completados para poder guardar el contenido.

### 20.1.10. Historia - LIBRERIA_001

**Nombre corto:** Conectar librería con aplicación en el CMS.

**Descripción:** Como usuario de la librería deseo poder conectar mi página web con mi aplicación administrada en el CMS.

**Criterios de aceptación:**

- Si el usuario proporciona un identificador y clave de aplicación válidos la librería debe poder conectarse con la aplicación en el CMS y obtener el token de autenticación correspondiente.

### 20.1.11. Historia - LIBRERIA_002

**Nombre corto:** Mostrar lista de contenidos.

**Descripción:** Como usuario de la librería deseo obtener y mostrar en mi web un listado de contenidos que haya cargado para una entidad.

**Criterios de aceptación:**

- La librería debe poder conectarse a la aplicación en el CMS y obtener los datos para la entidad que indique el usuario por medio del identificador de la entidad.
- La librería debe insertar los datos obtenidos en la sección que indique el usuario dentro de su web.
- La librería debe utilizar el contenido HTML de la sección como template para renderizar el contenido recibido, repitiendo el mismo una vez por cada contenido recibido.
- Dentro de cada template el usuario podrá indicar qué atributos mostrar utilizando sus identificadores.

- El usuario puede definir en el template uno, varios o todos los atributos pertenecientes a una entidad y la librería debe mostrar sólo los contenidos para los atributos indicados por el usuario.
- Por cada contenido devuelto la librería debe en todos los casos traer siempre el identificador de dicho contenido.

### 20.1.12. Historia - LIBRERIA_003

**Nombre corto:** Mostrar detalle de contenido.

**Descripción:** Como usuario de la librería deseo obtener y mostrar en mi web un contenido específico que haya cargado para una entidad.

**Criterios de aceptación:**

- La librería debe poder conectarse a la aplicación en el CMS y obtener los datos vinculados al contenido indicado por el usuario mediante su identificador.
- La librería debe insertar los datos obtenidos en la sección que indique el usuario dentro de su web.
- La librería debe utilizar el contenido HTML de la sección como template para renderizar el contenido recibido.
- Dentro del template el usuario podrá indicar qué atributos mostrar utilizando sus identificadores.
- El usuario puede definir en el template uno, varios o todos los atributos pertenecientes a una entidad y la librería debe mostrar sólo los contenidos para los atributos indicados por el usuario.

## 20.2. Organización de los Sprints

Las historias de usuario descriptas anteriormente serán organizadas en Sprints y ordenadas de acuerdo a su prioridad y a la dependencia que existe entre las mismas para evitar que se produzcan bloqueos por funcionalidades requeridas que aún no fueron desarrolladas.

Tabla 7

*Organización de las historias de usuario en Sprints*

| Identificador | Nombre | Prioridad | Complejidad | Sprint |
|---|---|---|---|---|
| **CMS_001** | Login usuario CMS | Media | Baja | 1 |
| **CMS_002** | Ingreso a panel de aplicación | Media | Baja | 2 |
| **CMS_003** | Crear aplicación | Alta | Media | 2 |
| **CMS_004** | Administrar datos de la aplicación | Baja | Baja | 2 |
| **CMS_005** | Administrar entidades de la aplicación | Alta | Alta | 2 |
| **CMS_006** | Administrar contenido por entidad | Alta | Alta | 3 |
| **LIBRERIA_001** | Conectar librería con aplicación en el CMS | Alta | Media | 4 |
| **LIBRERIA_002** | Mostrar lista de contenidos | Alta | Alta | 4 |
| **LIBRERIA_003** | Mostrar detalle de contenido | Media | Alta | 4 |
| **BACKEND_001** | Login administrador | Baja | Baja | 5 |
| **BACKEND_002** | Administrar cuentas de usuario | Baja | Baja | 5 |
| **BACKEND_003** | Administrar aplicaciones | Baja | Baja | 5 |

En la tabla anterior aparecen organizadas las historias en Sprints. Cuántas historias entran por Sprint es una decisión que se tomó en base a experiencia previa. Vemos que para desarrollar el producto necesitaremos un total de 5 iteraciones.

Cada Sprint tendrá una duración de 2 semanas de trabajo en las que me comprometo a dedicar, en promedio, unas 20 horas semanales. Al final de cada Sprint se pretende que estén cerradas las funcionalidades descriptas en las historias de usuario de ese Sprint y de cualquier otro Sprint anterior que aún no se haya podido cerrar o bien se hayan detectado errores que deban corregirse.

### 20.3. Sprint 0

Lo planificado anteriormente responde directamente a las historias de usuario. Pero para llevar adelante el proyecto se incluirá un Sprint especial (el Sprint 0) el cual será destinado a la consecución de las siguientes tareas:

- **Analizar el problema:** es necesario comprender lo que se pretende desarrollar, identificar las entidades, los usuarios y sus roles, los distintos subsistemas participantes y el contexto en el que se ejecutará cada uno;
- **Dar un nombre al producto:** resulta importante poder identificar al producto con alguna palabra o frase. Esto ayudará a facilitar la comunicación pero también a la hora de decidir, por ejemplo, el nombre de un repositorio o base de datos vinculados al proyecto.
- **Definir arquitectura de la solución:** permitirá tener una visión general de los subsistemas involucrados, componentes y sus comunicaciones;
- **Definir arquitectura de la base de datos:** permitirá tener una primera aproximación de las entidades involucradas y sus relaciones;
- **Crear base de datos y configurar servidores:** Es necesario contar con los ambiente configurados antes de comenzar el desarrollo;
- **Crear repositorios de código:** Necesario para garantizar que no se pierda nada a nivel de código a lo largo del proceso de desarrollo;
- **Diseñar prototipos de las interfaces:** Servirán como base para el desarrollo de las interfaces finales y además ayudarán a la tarea de análisis.

### 20.3.1. Análisis del problema

El producto CMS que se pretende construir se enfocará principalmente en la gestión de contenido. Si bien esto puede resultar evidente para el tipo de software que se está por construir lo cierto es que, como ya se describió anteriormente, muchas soluciones actuales sobrepasaron estas fronteras para no sólo gestionar contenido sino también estilos, templates, componentes, etc.

Esto provoca una separación poco clara entre los componentes principales sobre los cuáles se centra la arquitectura de todo software: el modelo (los datos), la vista (el template o contenedor, la presentación) y el controlador (vincula los otros dos componentes y provee lógica de negocio adicional).

#### 20.3.1.1. Modelo de datos

El sistema a construir se centrará entonces en la gestión del modelo de datos, requiriendo:

- Poder definir las distintas estructuras de datos dinámicas (definidas por el usuario) que se almacenarán;
- Almacenar los datos propiamente dichos.

Como ya se enunció en el apartado metodológico, en este el proyecto se utilizará una base de datos de tipo NoSQL orientada a documentos lo que permitirá que podamos almacenar estructuras dinámicas y flexibles.

En dicha base de datos, entonces, se almacenarán las entidades que cree el usuario junto con los atributos y tipos de datos que les haya asignado.

Otra ventaja del uso de documentos es que permiten almacenar la información tal cual es definida por el usuario (incluso con relaciones anidadas). De esta manera se evita tiempo de procesamiento (tanto al crear como también al recuperar los datos) en comparación con una alternativa en la que se utilice un motor SQL relacional.

#### 20.3.1.2. SaaS

La solución que se pretende construir busca ser fácil y rápida de usar y aprender por parte del usuario final. Uno de los puntos que se persigue es evitar que el usuario tenga que configurar servidores y gestionar bases de datos. Por esta razón se considera que un producto de software que funcione como servicio en la nube podría ser una buena alternativa al respecto.

Ahora bien, esto conlleva cierta complejidad que deberá tenerse en cuenta a la hora de definir la arquitectura. Por ejemplo, al gestionar en nuestro sistema la información de varios clientes tendremos que tener en cuenta lo siguiente:

- **Seguridad:** cada cliente sólo debe poder acceder a su propio contenido;
- **Concurrencia:** las operaciones que realice un cliente no debieran afectar (al menos no en forma directa) al resto de los clientes que utilicen el servicio;
- **Escalabilidad:** si se considera una solución que pueda ser escalable a cientos o miles de clientes entonces debemos establecer mecanismos que permitan crecer de manera horizontal, es decir, poder distribuir la información entre distintos servidores de ser necesario;
- **Encapsulamiento:** tiene también un poco que ver con lo ya mencionado de seguridad y concurrencia, pero ahora destacando que cualquier error, problema o inconveniente con la aplicación de un cliente no debiera afectar al resto de los usuarios de otras aplicaciones.

#### 20.3.1.3. Renderizado de los datos

Otro punto importante que se abordará es el mecanismo que se proveerá para integrar los contenidos almacenados en el CMS en la web del usuario. Aquí es donde entra en juego la ya mencionada Librería Cliente, cuyo principal objetivo será comunicarse con la aplicación CMS para obtener los datos de una entidad particular y mostrarlos embebidos en la plantilla definida.

Dicha librería debería ser capaz de funcionar en cualquier sitio web, por lo que debe contemplarse el uso sólo de tecnologías estándar como HTML, CSS

y JavaScript. Esto posibilitará lograr el objetivo de integración sin importar el lenguaje de programación o plataforma de servidor utilizada.

Al ser el usuario quien defina la estructura HTML del template le daremos más facilidad y flexibilidad para construir los diseños que desee. Esto también permitirá que el usuario maneje su propia convención de nombres o nomenclatura a la hora de, por ejemplo, definir reglas de estilos en sus hojas CSS.

Por último, al delegar la responsabilidad de renderizar las vistas al navegador del visitante estamos logrando distribuir este esfuerzo de procesamiento. En una solución CMS convencional es el mismo servidor web quien debe procesar las plantillas para devolverla luego con los datos embebidos.

#### 20.3.1.4. Coexistencia con otros sistemas

Otra de las falencias de los sistemas CMS existentes es que son muy cerrados, no sólo en cuanto a lenguaje de programación o plataforma, sino también a la hora de permitir la integración con otros sistemas.

Es por eso que debemos facilitar, o por lo menos no limitar ni entorpecer, la comunicación y coexistencia entre sistemas.

Una forma de lograr esto es brindando una API que pueda ser utilizada por los usuarios para consultar datos de su aplicación.

### 20.3.2. Nombre del producto

El producto recibirá el nombre PranaCMS.

Prana es una palabra en sánscrito que hace referencia al aire que se inspira y se lo relaciona directamente con la energía vital.

La elección de este nombre responde a las siguientes características:

- Es corto y fácil de recordar;
- Su pronunciación es similar en varios idiomas lo que facilitaría su difusión;
- Por su relación al aire y al yoga da sensación de pureza, fortaleza y equilibrio. Estos conceptos pueden usarse como análogos al producto CMS que se desarrollará.

### 20.3.3. Arquitectura general de la solución

De acuerdo a lo analizado previamente (y a lo largo del trabajo) se derivan varias pautas y conclusiones:

- Se requiere como mínimo el desarrollo de 2 sistemas: uno que le permita al usuario gestionar las entidades y datos de su aplicación, y otro que le permita mostrar dichos datos en cualquier aplicación web;
- Debe existir alguna capa de comunicación y transferencia de información entre los 2 sistemas antes mencionados;
- Ya que cada cliente podrá gestionar sus propias estructuras e información será necesario proveer algún mecanismo flexible para tal fin, a la vez que se deberá garantizar cierto nivel de seguridad para no exponer los datos de un cliente a otro que no corresponda;
- Existe además otro tipo de estructuras que pueden ser comunes entre todos los clientes y que tienen un carácter más administrativo, como por ejemplo la gestión de aplicaciones por cliente y la gestión de usuarios por aplicación.

Entonces, la arquitectura estará compuesta de la siguiente forma:

- Existirá una base de datos general que guardará información común a todos los clientes, como por ejemplo las aplicaciones y usuarios. Para almacenar dichos datos se optará por una base de datos relacional como lo es MySQL.
- Cada cliente contará con su propia instancia de base de datos. Esto trae aparejado algunas ventajas inherentes, como por ejemplo:
  - **Seguridad:** cada cliente sólo podrá conectarse a su propia base de datos mediante usuario y contraseña;
  - **Encapsulamiento:** cada cliente sólo podrá acceder y almacenar allí los datos de su propia aplicación;
  - **Concurrencia:** los motores de base de datos suelen permitir accesos concurrentes a nivel de lectura, pero bloquean las tablas y registros durante su modificación. Entonces, de esta manera el trabajo de un cliente sobre sus estructuras de datos no afectaran al rendimiento del resto de los usuarios de otras aplicaciones.
  - **Escalabilidad:** al no concentrar los datos de todas las aplicaciones clientes en una única base de datos será más fácil la tarea de escalar horizontalmente en un futuro distribuyendo las bases de datos enteras entre distintos servidores.
- Las bases de datos de los clientes utilizarán un motor no relacional como lo es MongoDB. Las colecciones en estas bases de datos tendrán referencias foráneas a registros en las tablas de la base de datos relacional;
- El acceso a los datos estará centralizado en una capa de servicio web. Puntualmente se trabajará con APIs REST que permiten comunicación por HTTP. Para el desarrollo de esta aplicación se utilizará Sails.js.
- Estos servicios web serán consumidos por al menos dos tipos de aplicaciones:
  - La aplicación Backend es a la que accederán los usuarios del CMS para gestionar los datos de sus aplicaciones. Ésta será una aplicación común para todos los clientes y podrán acceder mediante usuario y contraseña.

Esta aplicación estará principalmente desarrollada utilizando el framework Angular.

- La Librería Cliente será la que se encargue de obtener los contenidos de una aplicación determinada. Para determinar la aplicación a la que se conecta se deberá proveer de un identificador y clave, el cual será generado automáticamente para cada aplicación. Esta librería debe ser incluida y configurada en la web que se desee conectar con el CMS. Esta librería estará desarrollada utilizando como base el framework Riot.js.

Esta arquitectura trae aparejada las siguientes ventajas:

- Si bien en un principio es probable que toda la solución esté dispuesta en un único servidor, la separación en capas permite que a futuro cada una de ellas pueda ser ubicada en un servidor diferente;
- A su vez, como ya se mencionó, la capa de datos también puede ser distribuidas entre varios servidores para optimizar la performance;
- La capa de datos sólo es accesible mediante la capa de servicios. Esto permite centralizar el acceso a los datos en un único punto lo que da mayor control y seguridad. Además, las aplicaciones no tendrán acceso directo a los datos lo que, además de ser más seguro, posibilita el cambio de cualquier componente de la solución sin afectar al resto del sistema;
- La elección tanto de un motor de base de datos relacional como también de otro no relacional permitirá sacar mayor provecho a cada una de dichas tecnologías, explotando los beneficios para los cuáles fueron pensadas: uniformidad, integridad y consistencia para la base de datos general y flexibilidad y velocidad para la escritura y acceso de datos de estructuras complejas para el caso de las bases de datos no relacionales.
- El uso de node.js como plataforma de servidor resulta útil en este contexto ya que permite comunicación asincrónica con cada uno de los nodos a los que se conecta permitiendo así, por ejemplo, comunicación con las dos bases de datos al mismo tiempo sin bloquear el proceso.
- La elección de tecnologías basadas en JavaScript permitirá mantener el mismo lenguaje común para todo el desarrollo e incluso la posibilidad de compartir o reutilizar código entre las aplicaciones.

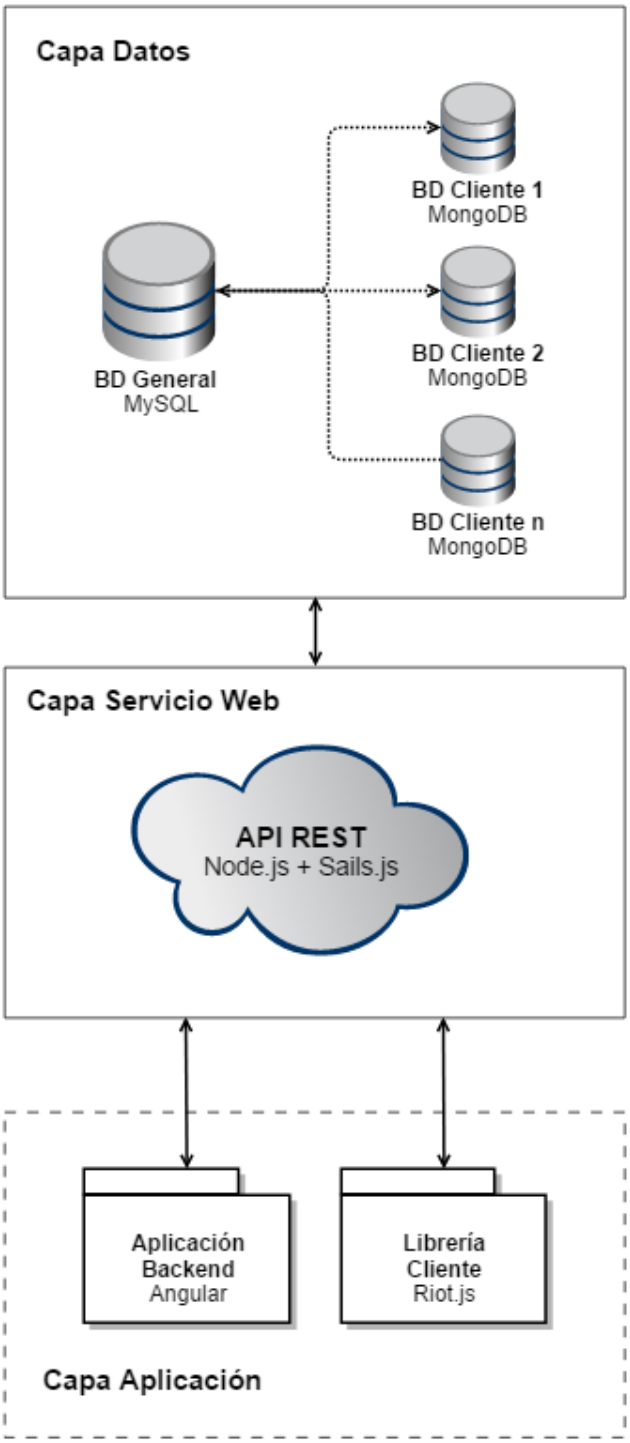



*Figura 11: Arquitectura general de la solución. Fuente: Elaboración propia*

### 20.3.4. Arquitectura de la base de datos

En el siguiente gráfico se presenta el diagrama general de la base de datos relacional:

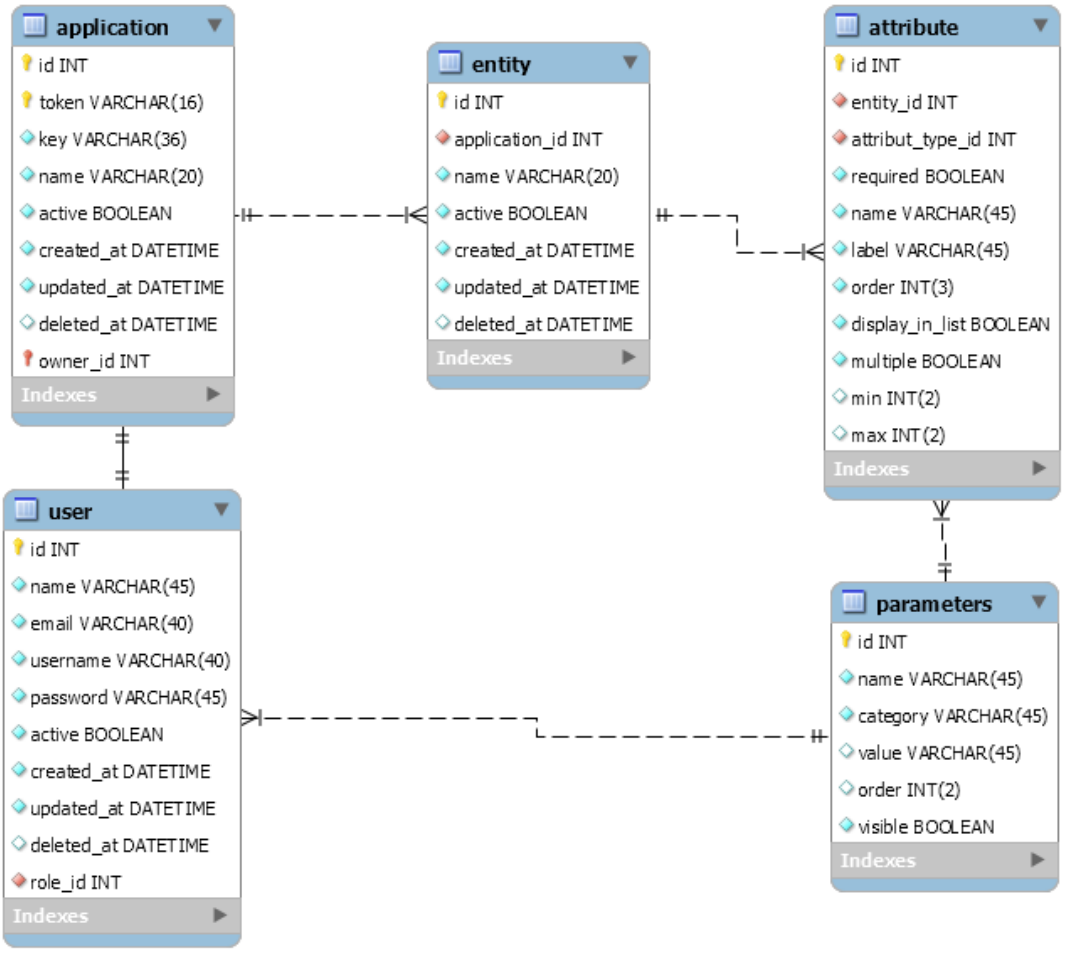



*Figura 12: Arquitectura de la base de datos relacional. Fuente: Elaboración propia*

A continuación se describen las responsabilidades de cada tabla:

- **Application:** representa la aplicación creada por el usuario. Contiene un identificador único público (token) y una clave privada (key). Guarda una referencia al usuario creador.
- **User:** almacena la información de los usuarios registrados. Un usuario puede ser dueño de más de una aplicación. Por el momento no se contempla la funcionalidad de gestionar más de un usuario por aplicación. Un usuario puede pertenecer al rol ADMINISTRADOR_SISTEMA o al rol USUARIO_CMS. El rol está referenciado a una entrada en la tabla de parámetros de configuración.
- **Entity:** representa a una entidad creada por el usuario para una aplicación. Ésta puede ser identificada por un nombre y tendrá relación a uno o más registros de la tabla de atributos.
- **Attribute:** representa un campo de una entidad. Un campo tiene un nombre que lo identifica. Además define otras características que permitirán, por ejemplo, marcar el campo como requerido, mostrarlo en el orden deseado o

definir si este campo se utilizará para mostrar el contenido en la grilla. También se podrá indicar si el mismo permite la asociación de uno o más valores independientes. Por último tenemos un campo que nos indica el tipo de atributo. Esto nos permitirá diferenciar, por ejemplo, entre un input de texto, de texto enriquecido, imagen, checkbox, entre otros. Estos tipos de campos básicos estarán tipificados en la tabla parámetros.

- **Parameters:** nos permitirá almacenar distintos valores de configuración que requiramos. Esta tabla genérica evita tener que generar tablas adicionales de parametrización, como por ejemplo para los roles o tipos de campo básicos. Esta tabla, como ya se mencionó, es de configuración por lo que los usuarios no podrán manipular sus valores, vienen pre-establecidos.

En estas estructuras se guardará todo lo necesario para que un usuario pueda administrar las entidades de su aplicación y definir sus campos.

Para almacenar el contenido que genere el usuario para cada entidad, sin embargo, se utilizarán bases de datos de tipo no relacional.

- Se creará una base de datos en el motor MongoDB por cada aplicación. Tanto el nombre como la contraseña para conectarse serán generados automáticamente y no podrán ser editados por el usuario.
- Por cada entidad que genere el usuario se creará una colección en esta base de datos. Se utilizará el identificador de la entidad para autogenerar el nombre de la colección.
- Ya que la base de datos es no relacional y orientada a documentos, cada vez que el usuario genere un nuevo contenido para una entidad se generará un nuevo documento en la colección correspondiente. Los datos almacenados en dicho documento se corresponden a los atributos definidos para esa entidad.

En el Anexo A se presenta un ejemplo que ilustra cómo se utilizarán algunas de las estructuras de datos antes presentadas.

### 20.3.5. Repositorios de código

Como ya se documentó en el apartado metodológico para el presente proyecto se utilizará GIT como sistema de versionado.

Se crearán 3 repositorios diferentes:

- prana-api: para el versionado de la API Rest;
- prana-admin: para el versionado de la aplicación Backend;
- prana-client: para el versionado de la Librería Cliente.

### 20.3.6. Interfaces gráficas

A continuación se presenta una serie de interfaces gráficas que serán utilizadas como entrada a lo largo del proceso de desarrollo. Se aclara que las mismas

servirán sólo a modo de guía pero no determinarán necesariamente el diseño final de la aplicación.

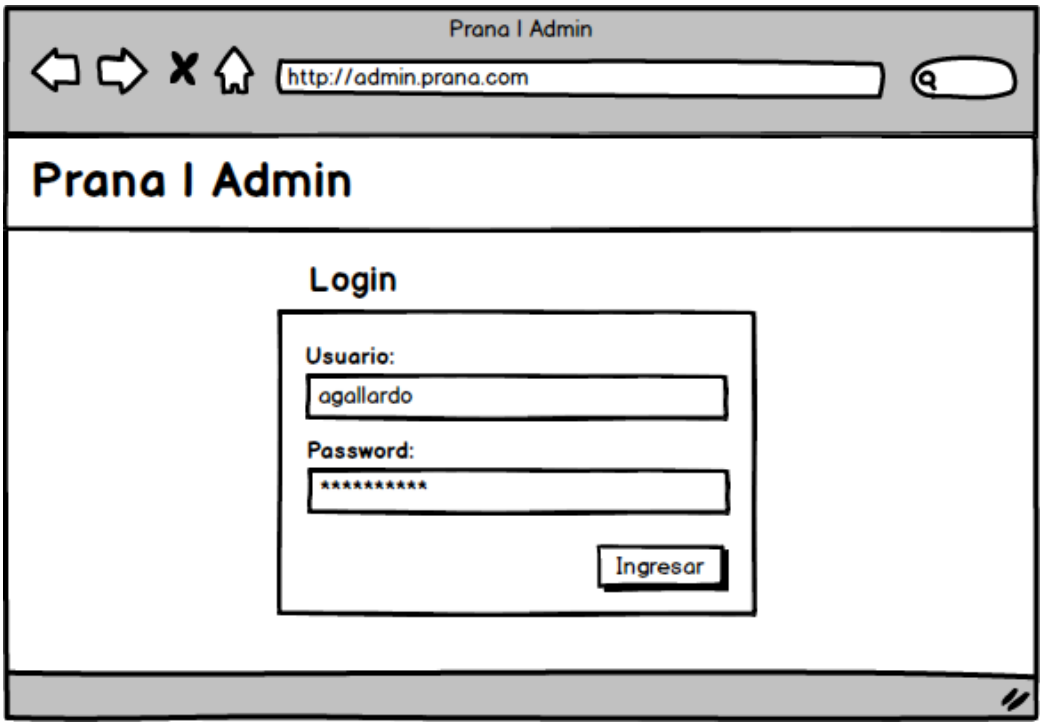



*Figura 13: Boceto para la pantalla de Login de usuarios. Fuente: Elaboración propia*


*Figura 14: Boceto para la pantalla de listado de aplicaciones (rol administrador). Fuente: Elaboración propia*

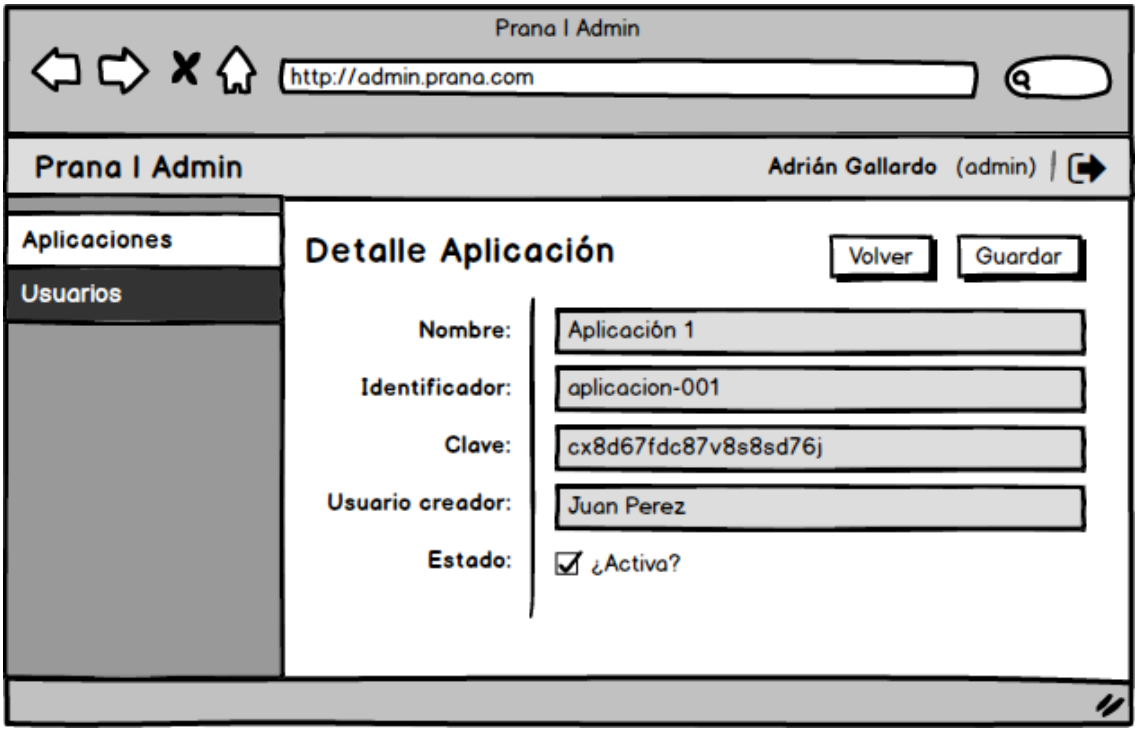



*Figura 15: Boceto para la pantalla de visualización de datos de una aplicación (rol administrador). Fuente: Elaboración propia*

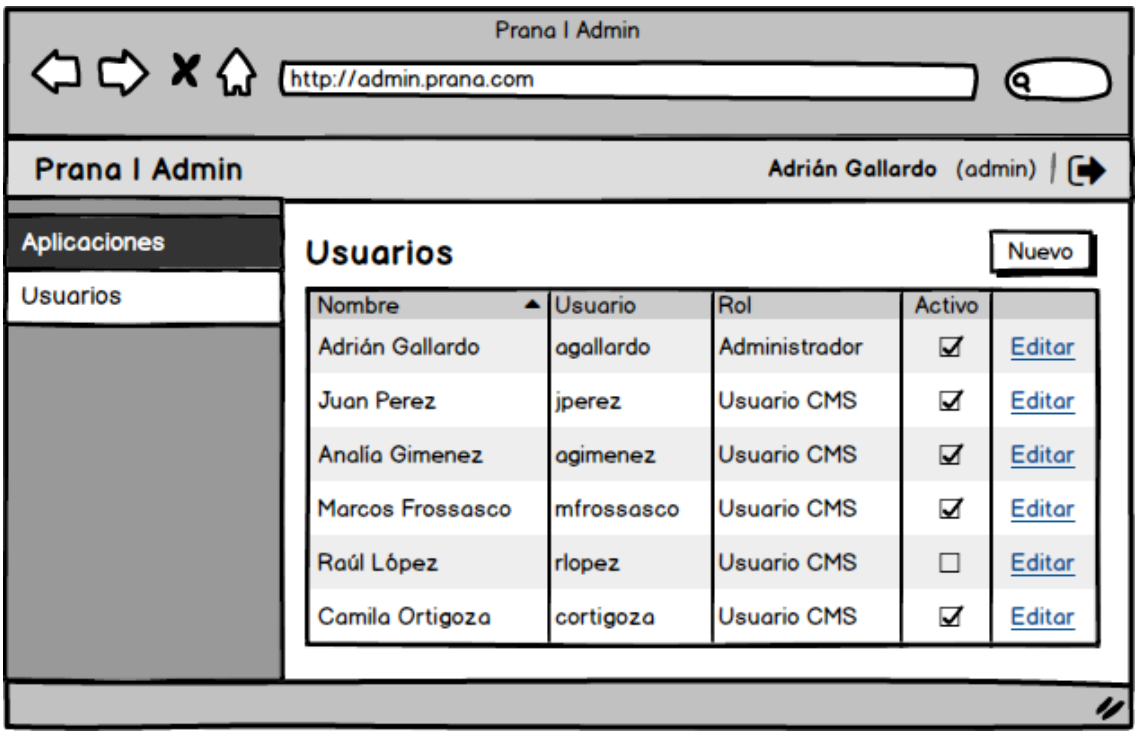



*Figura 16: Boceto para la pantalla de listado de usuarios del sistema (rol administrador). Fuente: Elaboración propia*

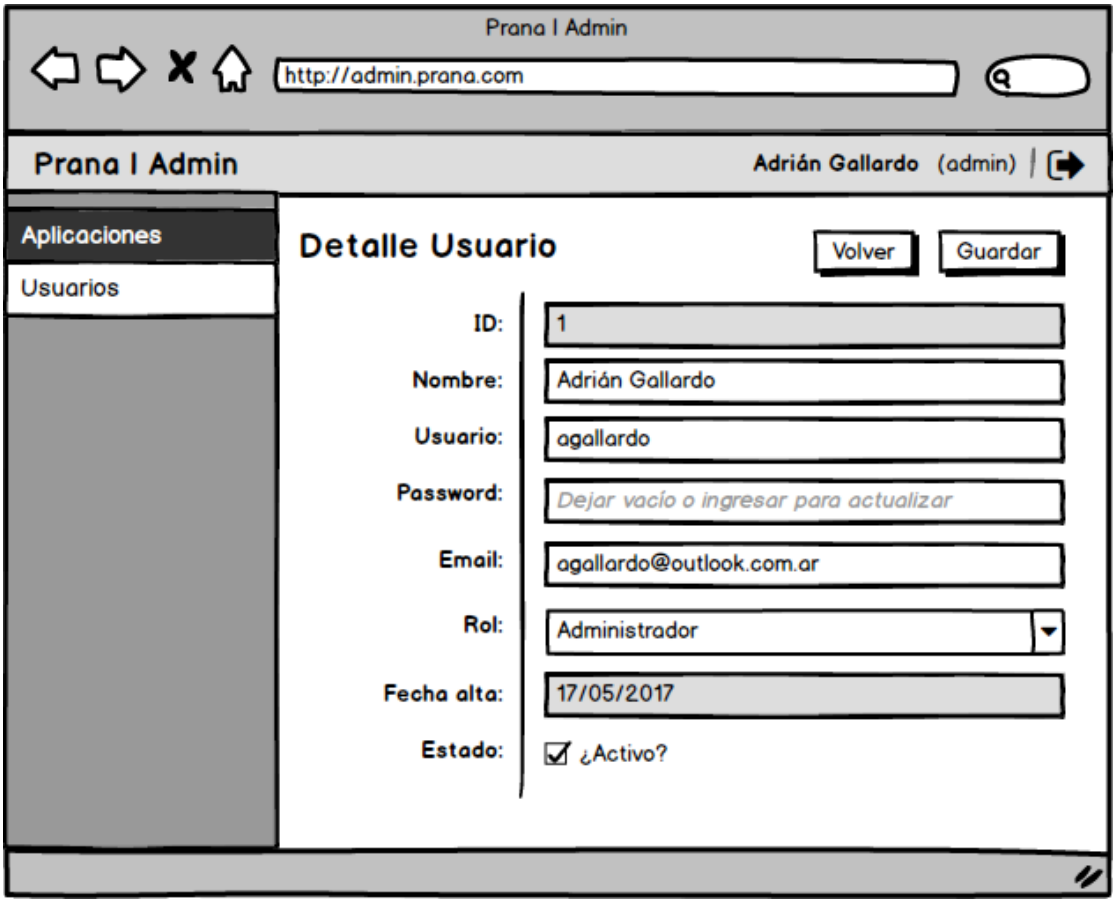



*Figura 17: Boceto para la pantalla de edición de datos de un usuario (rol administrador). Fuente: Elaboración propia*

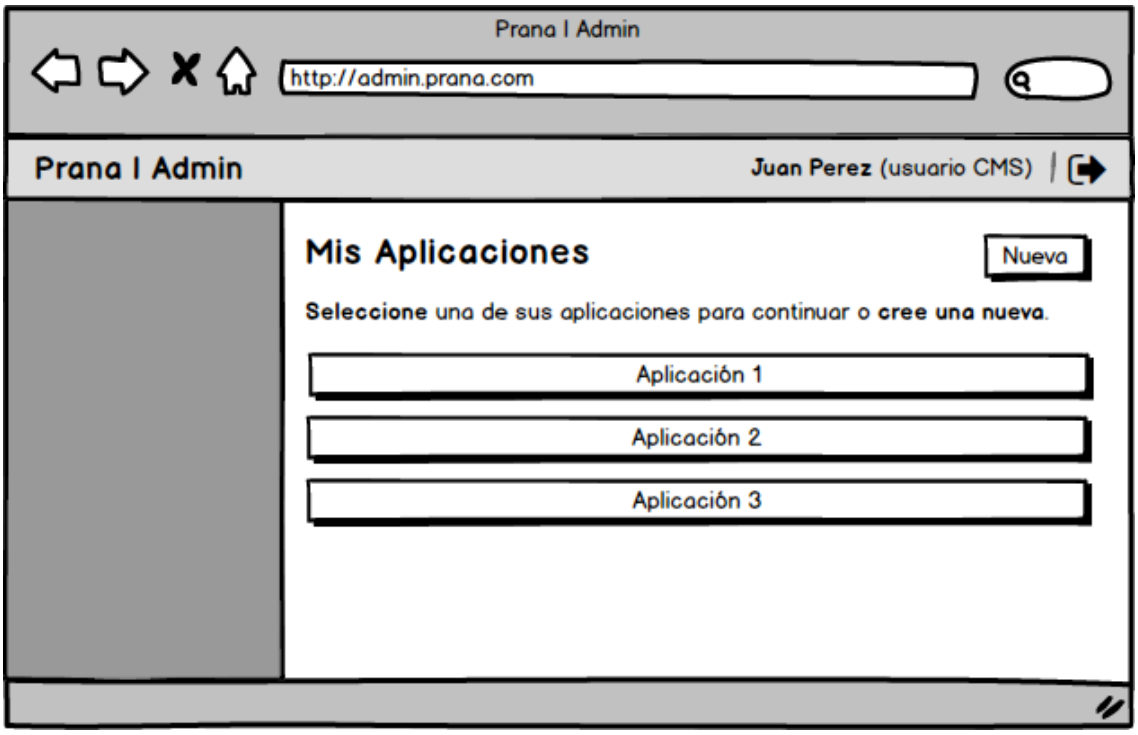



*Figura 18: Boceto para la pantalla de listado de aplicaciones de un usuario (rol usuario cms). Fuente: Elaboración propia*

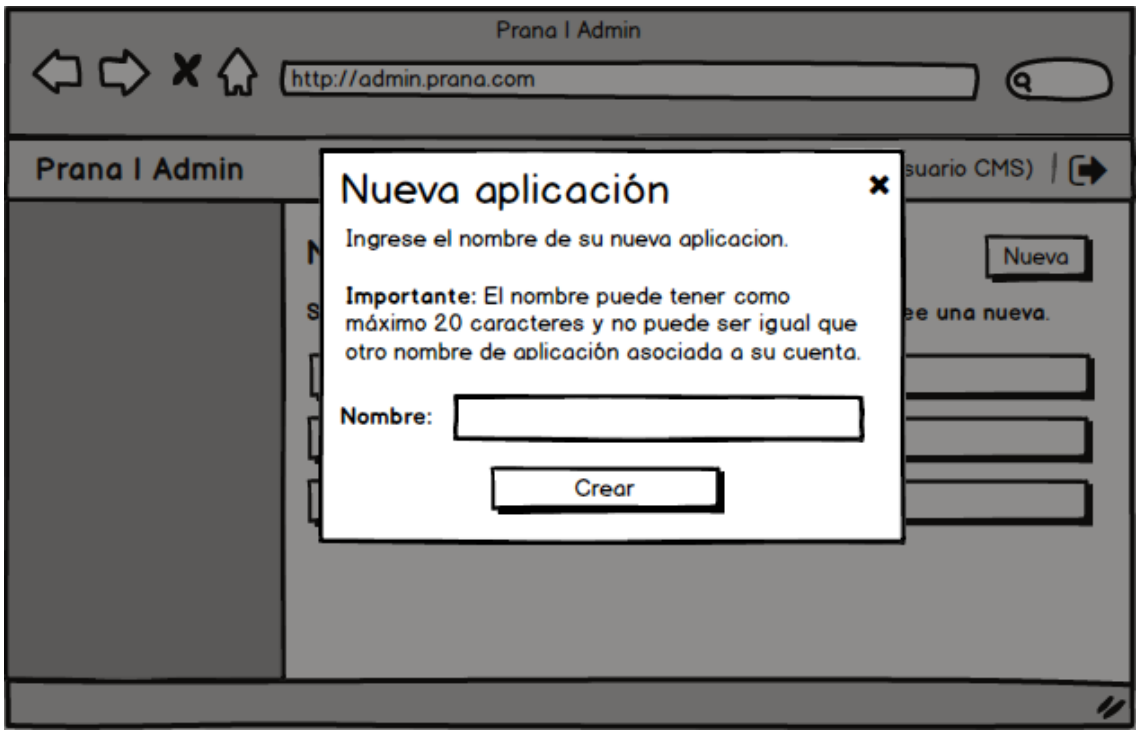



*Figura 19: Boceto para el formulario de creación de nueva aplicación (rol usuario cms). Fuente: Elaboración propia*

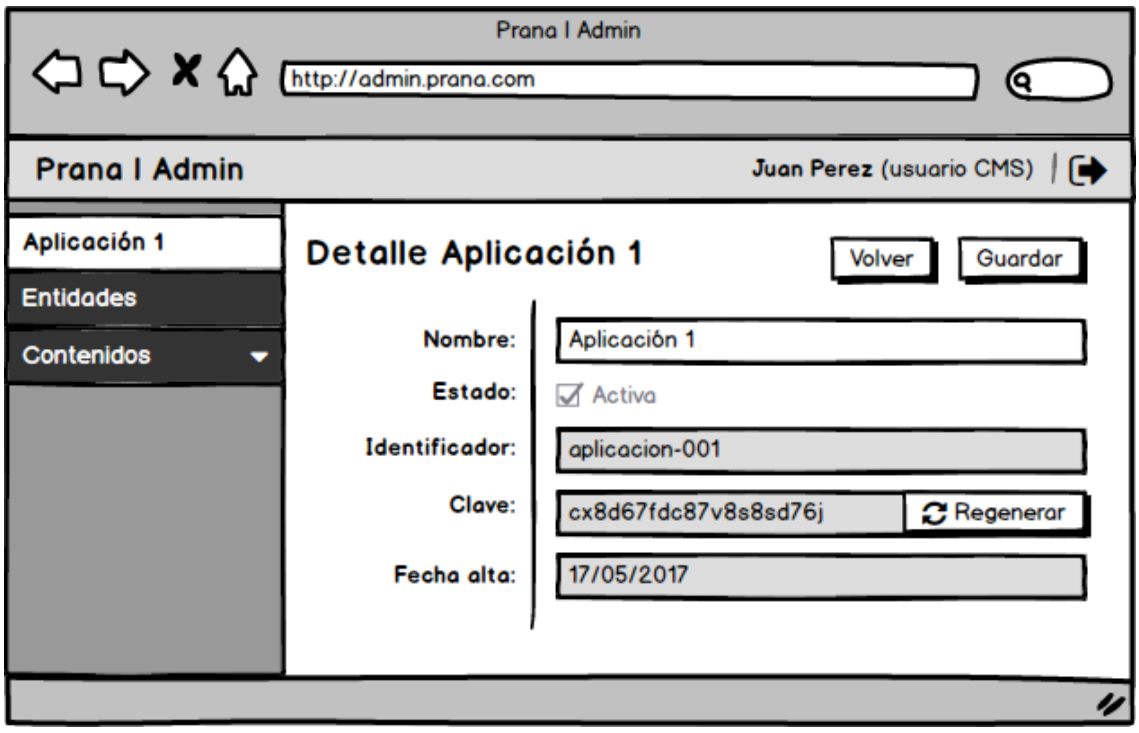



*Figura 20: Boceto para la pantalla de visualización de datos de una aplicación (rol usuario cms). Fuente: Elaboración propia*

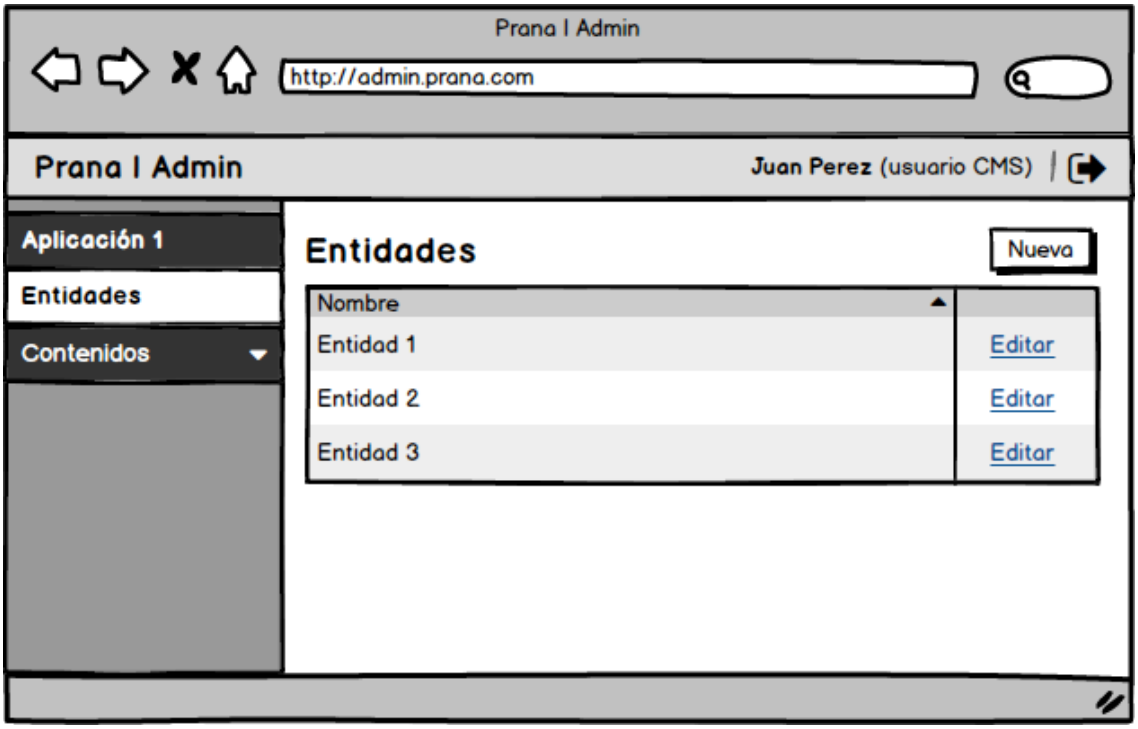



*Figura 21: Boceto para la pantalla de listado de entidades de una aplicación (rol usuario cms). Fuente: Elaboración propia*

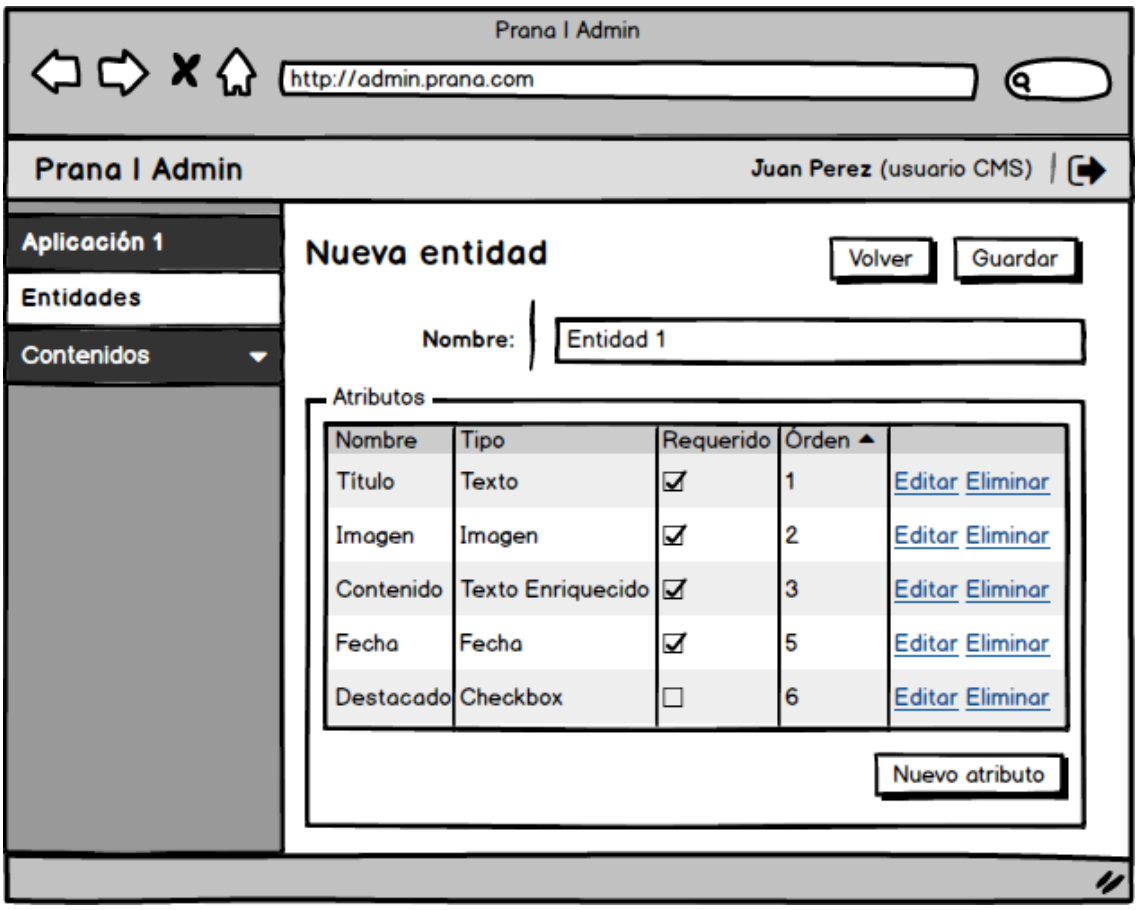



*Figura 22: Boceto para la pantalla de creación de una nueva entidad (rol usuario cms). Fuente: Elaboración propia*


*Figura 23: Boceto para el formulario de alta de un nuevo atributo para una entidad (rol usuario cms). Fuente: Elaboración propia*

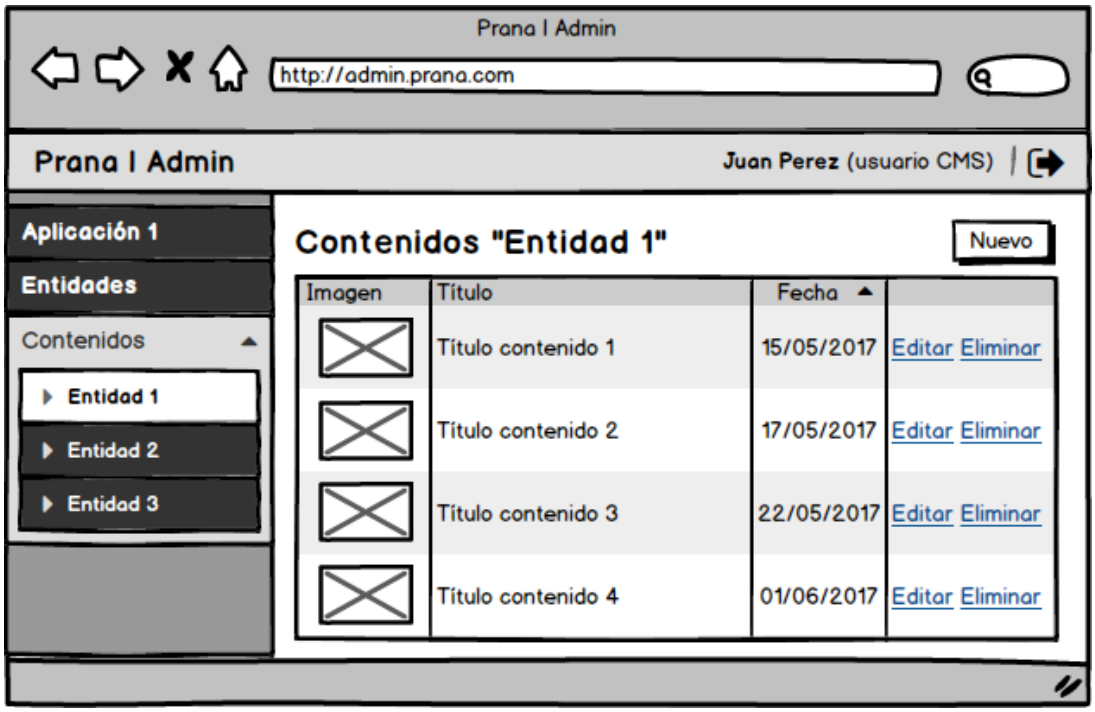



*Figura 24: Boceto para la pantalla de listado de contenidos de una entidad (rol usuario cms). Fuente: Elaboración propia*

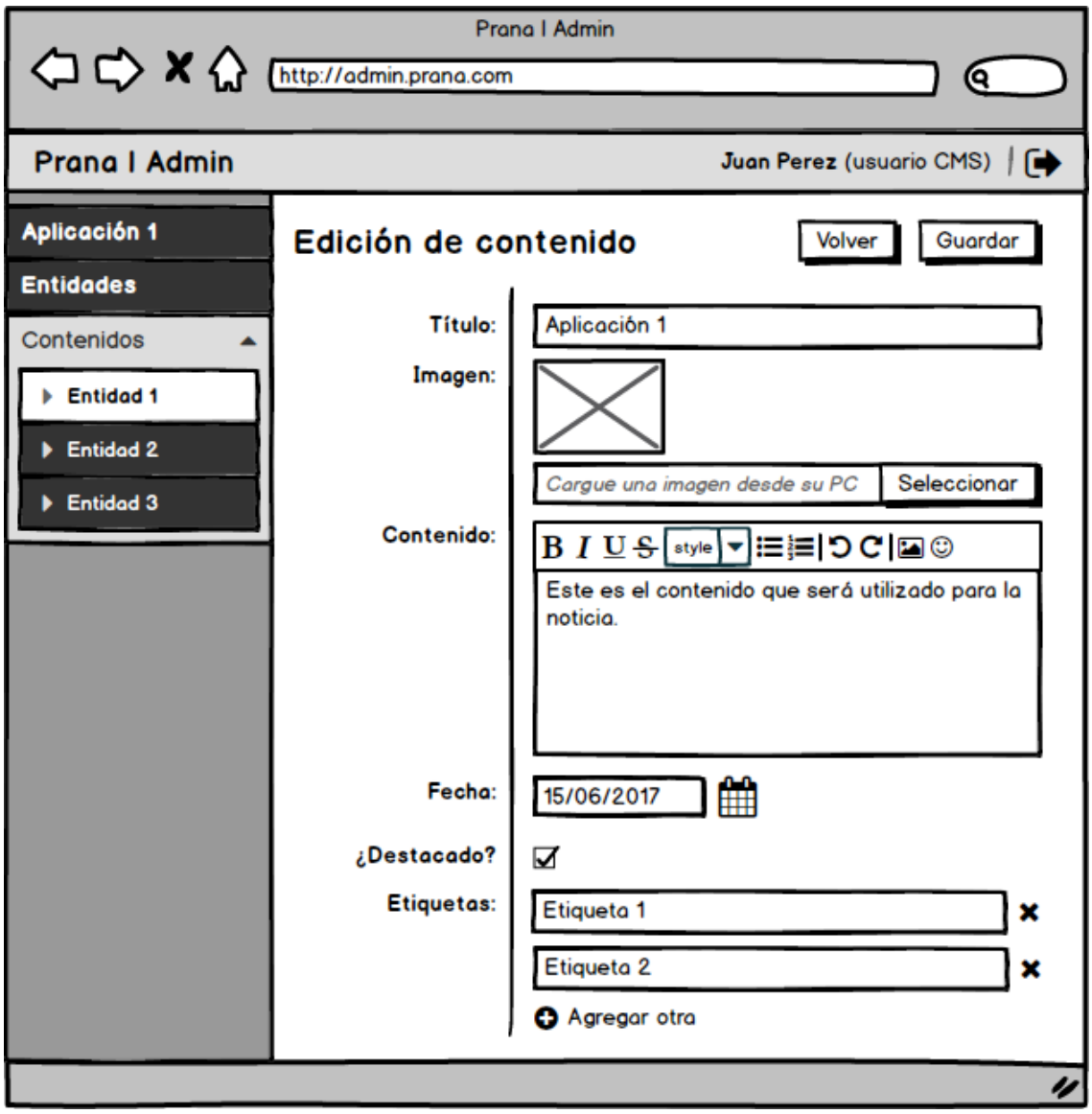



*Figura 25: Boceto para la pantalla de creación de nuevo contenido para una entidad (rol usuario cms). Fuente: Elaboración propia*

### 20.4. Sprint 1

En este primer Sprint se trabajó sobre la creación y configuración de los proyectos y entornos de trabajo, y se abordó la primera historia de usuario relacionada al Login en el CMS.

El Sprint Backlog sobre el que se trabajo es el siguiente:

Tabla 8
*Sprint Backlog del Sprint 1*

| Historia / SubTareas | Estimación original | Estimación actualizada | Sobreestimado Subestimado | Tiempo Trabajado |
|---|---|---|---|---|
| **Generales Inicio Proyecto** | | | | |
| Configurar entorno de trabajo | 4 | 4 | 0 | 4 |
| Maquetación y estilos generales de la aplicación | 10 | 12 | -2 | 12 |
| **CMS_001: Login usuario CMS** | | | | |
| Generar modelo User | 1 | 1 | 0 | 1 |
| Generar API CRUD para User | 1 | 2 | -1 | 2 |
| Generar API para el Login | 2 | 2 | 0 | 2 |
| Generar y devolver JWT (JSON Web Token) | 6 | 8 | -2 | 8 |
| Validar User logueado mediante su JWT | 2 | 2 | 0 | 2 |
| Generar modelo Parameters | 1 | 1 | 0 | 1 |
| Generar API CRUD para Parameters | 2 | 1 | 1 | 1 |
| Maquetar pantalla Login usuario | 2 | 2 | 0 | 2 |
| Consumir servicio de login y validar datos del usuario | 2 | 6 | -4 | 6 |
| Restringir acceso a sección de usuarios logueados | 1 | 1 | 0 | 1 |
| Maquetar pantalla de Landing para usuarios logueados | 1 | 1 | 0 | 1 |
| **TOTAL** | 35 | 43 | -8 | 43 |

Como en todo proyecto de Software es esperable que existan diferencias entre lo proyectado y lo trabajado. En este caso hay unas 8 horas sub-estimadas (completar el Sprint tomó 8hs más de lo previsto).

Éste es un dato útil que ayudará a ajustar futuras estimaciones. Haciendo un análisis retrospectivo sobre este Sprint, las complicaciones principales que se encontraron fueron las siguientes:

- Complejidad técnica de implementar un mecanismo de autenticación de usuario basado en Tokens;
- Implementación de servicios REST utilizando el framework Angular;
- Tiempo de configuración y puesta a punto de la aplicación Angular y del Backend en Sails.js, por inexperiencia con ambas tecnologías.

### 20.4.1. Seguridad con JWT

Como ya se hizo referencia se implementó un protocolo de seguridad basado en Tokens.

Este mecanismo es de uso común cuando se trabajan con APIs REST ya que la aplicación cliente suele ser distinta de la aplicación de la capa de servicios, y por ende un mecanismo de seguridad basado en sesiones no es viable.

Se escogió un protocolo de autenticación simple basado en tokens que recibe el nombre de JWT de sus siglas en inglés JSON Web Tokens.

Este protocolo se volvió popular en el último tiempo por los siguientes motivos:

- El token es fácil de generar (existen muchas librerías de integración a distintos lenguajes y frameworks);
- Es de uso gratuito;
- Está codificado en base64 lo que es apto y seguro para su transporte por medio de HTTP;
- El mismo token contiene en su estructura 3 bloques de información:
    - Por un lado contiene información sobre el algoritmo de cifrado utilizado. Esto permite a la aplicación abstraerse de esos detalles técnicos ya que la librería al leer el token sabe automáticamente cómo procesarlo;
    - Posee datos clave que identifican al usuario, como su nombre, rol o id. Estos datos pueden ser libremente definidos por la aplicación;
    - El token además contiene en su cuerpo un hash de verificación que asegura que el token es íntegro y no fue adulterado, por lo tanto es de una fuente confiable.
- Contiene metadata que permite conocer por ejemplo cuándo se creó o cuándo caduca.

La secuencia de autenticación es la siguiente:

- El usuario ingresa a la aplicación y es dirigido al formulario de Login ya que no posee un token válido;
- El usuario completa los datos con su nombre de usuario y password y los envía a un servicio web para validarlos;
- Si el servidor web puede identificar al usuario entonces genera un token que representará inequívocamente a este usuario y lo devuelve como parte de su respuesta;
- La aplicación recibe y almacena este token en la PC del cliente;
- Luego, cuando un usuario intenta acceder a una sección segura la aplicación verificará que exista un token válido y vigente;
- Del mismo modo, cuando el usuario utilice alguna funcionalidad que invoca algún servicio web el token es enviado como parte de la cabecera del mensaje. Entonces, el servicio web verificará por un lado que el token sea válido, que no haya caducado y pertenezca a un usuario activo. Además, ya que el token contiene datos que permiten identificar al usuario es capaz de definir si el mismo posee privilegios o no para acceder a determinados recursos o bien si está autorizado o no a realizar alguna acción en particular.

La elección de este protocolo frente a otros más complejos como OAuth se debe a su versatilidad y rapidez para ser implementado ofreciendo un buen nivel de seguridad general en muy corto tiempo.

OAuth, por su naturaleza, resulta mucho más complejo de implementar. Si bien resulta ser un protocolo mucho más versátil y con mayores prestaciones, su complejidad técnica y tecnológica lo hace inviable para proyectos chicos, siendo más recomendado su uso en productos de gran envergadura.

### 20.4.2. Resultados del Sprint 1

A continuación se presentan las interfaces desarrolladas tras completar la primera iteración.

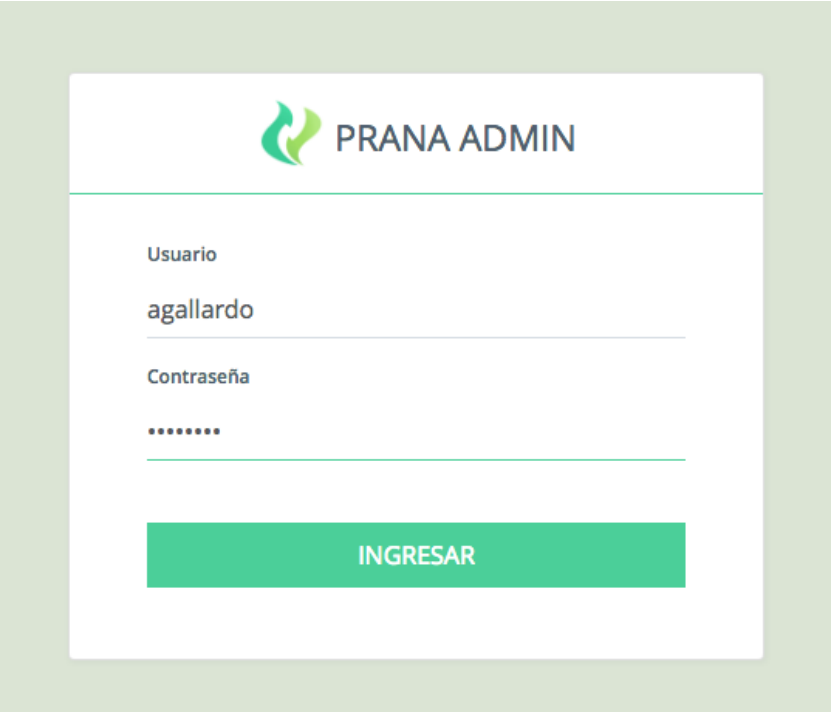



*Figura 26: Formulario de Login. Fuente: Elaboración propia*

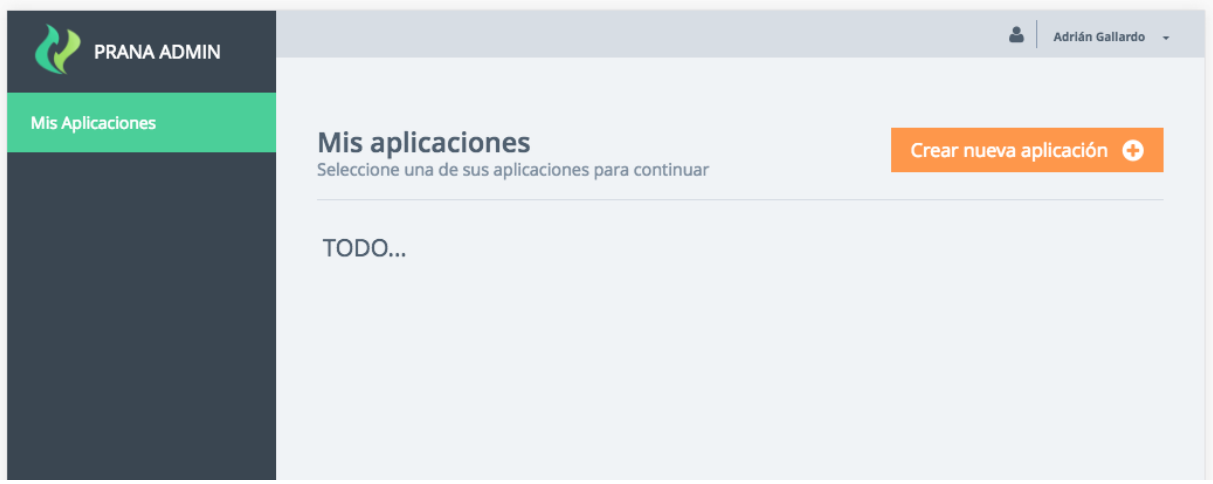
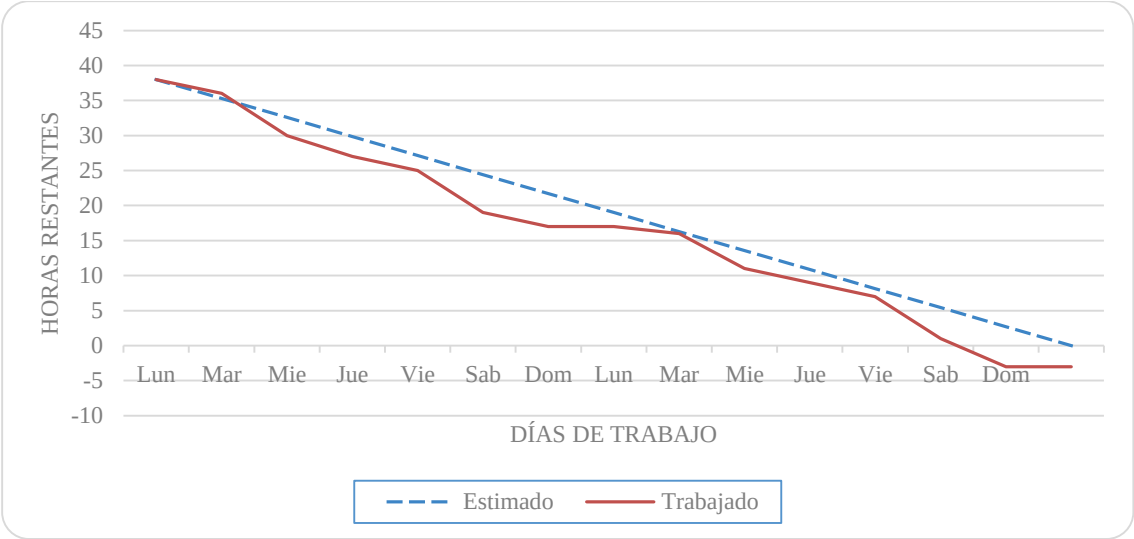



*Figura 27: Pantalla de inicio para el usuario logueado (rol usuario-cms).*
*Fuente: Elaboración propia*

### 20.4.3. Diagrama Burndown Sprint 1

El siguiente es el diagrama resultante tras haber completado el primer ciclo de trabajo.




La línea punteada descendente representa el avance ideal que día a día se debería conseguir para cumplimentar con lo pautado en tiempo y forma.

La línea continua representa el avance real sobre el proyecto de acuerdo a tiempo real trabajado y avance sobre las tareas del Sprint medidas en cantidad de horas trabajadas / cantidad de horas estimadas. Es decir, si una tarea está estimada en 2hs y llevo 1h de trabajo, entonces la misma se supone que está al 50% de avance. Si luego de trabajar esa hora se considera que la tarea puede llevar más o menos tiempo se puede ajustar la estimación para reflejar el progreso real o avance real sobre la misma.

Cuando esta línea contínua se encuentra por debajo de la línea punteada significa que el progreso alcanzado es mayor al previsto lo que hace pensar (al menos en ese instante temporal) que el proyecto va a terminarse en tiempo y forma.

Cuando la línea continua, en cambio, se encuentra por encima de la línea punteada significa que tenemos trabajo retrasado. Esto nos alerta y nos invita a tratar de revertir esta situación para poder llegar a tiempo y sin desvíos.

## 20.5. Sprint 2

En este Sprint se trabajó sobre la creación de aplicaciones, la edición de datos de una aplicación, el armado de menú con las opciones de cada aplicación, y la gestión de entidades por aplicación.

Tabla 9
*Sprint Backlog del Sprint 2*

| Historia / SubTareas | Estimación original | Estimación actualizada | Sobreestimado Subestimado | Tiempo Trabajado |
|---|---|---|---|---|
| **CMS_002: Ingreso a panel de aplicación** | | | | |
| Visualizar aplicaciones creadas por el usuario | 4 | 4 | 0 | 4 |
| Visualizar menú de acceso a entidades y contenidos de la aplicación seleccionada | 1 | 6 | -5 | 6 |
| Visualizar aplicaciones en estado inactivo | 2 | 1 | 1 | 1 |
| Obtener datos de la aplicación seleccionada | 2 | 2 | 0 | 2 |
| **CMS_003: Crear aplicación** | | | | |
| Alta de nueva aplicación | 4 | 4 | 0 | 4 |
| Generar base de datos NoSQL para la aplicación | 2 | 2 | 0 | 2 |
| **CMS_004: Administrar datos de la aplicación** | | | | |
| Visualizar y editar datos de aplicación seleccionada | 4 | 5 | -1 | 5 |
| **CMS_005: Administrar entidades de la aplicación** | | | | |
| Visualizar entidades de una aplicación | 2 | 4 | -2 | 4 |
| Alta y edición de los datos una entidad | 2 | 2 | 0 | 2 |
| Vincular uno o más atributos a una entidad | 2 | 8 | -6 | 8 |
| Eliminar atributo de una entidad | 2 | 2 | 0 | 2 |
| Configurar propiedades de cada atributo | 4 | 4 | 0 | 4 |
| Configurar orden de los atributos | 4 | 4 | 0 | 4 |
| **TOTAL** | 35 | 48 | -13 | 48 |

### 20.5.1. Resultados del Sprint 2

A continuación se presentan las interfaces desarrolladas tras completar la segunda iteración.

Antes que nada, se aclara que las entidades administradas por la aplicación recibirán el nombre de "formularios", ya que se considera que resultará un término más familiar para los usuarios. Del mismo modo los atributos de las entidades recibirán el nombre de "campos" de un formulario.

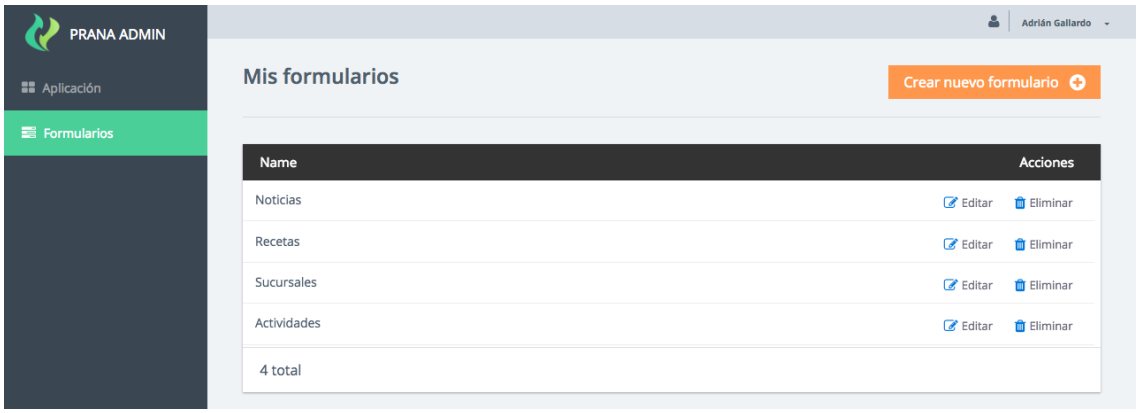



*Figura 28: Pantalla de listado de formularios de una aplicación (rol usuario-cms).*
*Fuente: Elaboración propia*

En la imagen anterior vemos una tabla con el listado de formularios creados por el usuario. Cada registro posee un botón que permite editar o eliminar el formulario.

Al hacer clic en el botón crear se mostrará la siguiente pantalla de alta de formulario:

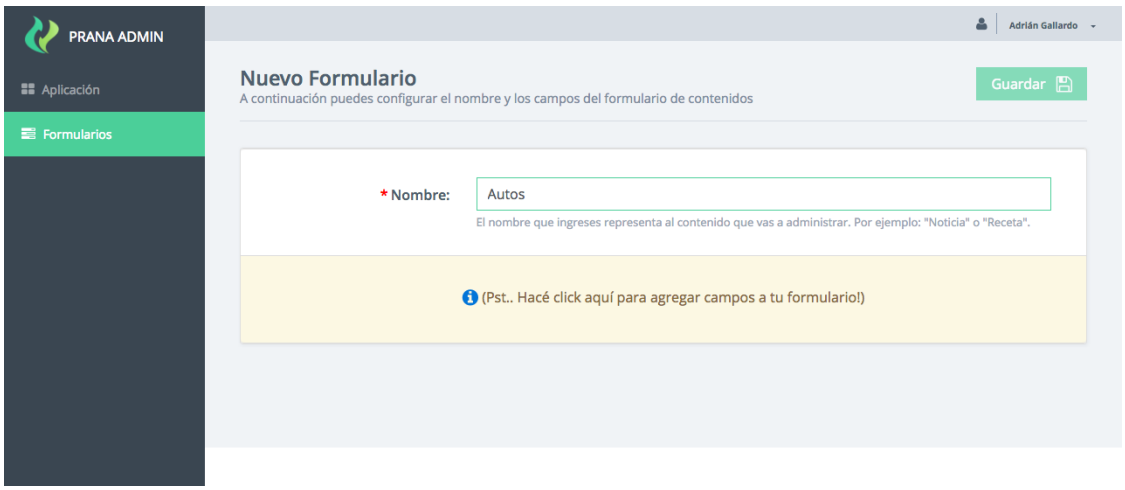



*Figura 29: Pantalla de creación de nuevo formulario (rol usuario-cms).*
*Fuente: Elaboración propia*

El formulario se compone de un nombre y de un conjunto de atributos. El nombre es obligatorio y sirve para que el usuario pueda identificarlo.

También se muestra un mensaje que indica al usuario que debe hacer clic para asociar un campo al formulario. El botón de guardar permanecerá inactivo hasta que no se configure un nombre y se asocie al menos un campo.

El alta y edición de los campos se harán por medio de un formulario que se muestra en una ventana flotante o modal.


*Figura 30: Formulario para cargar y asociar un nuevo campo a un formulario (rol usuario-cms). Fuente: Elaboración propia*

Se requiere el ingreso de un nombre y de un tipo de dato a seleccionar. En esta primera instancia las opciones posibles son: texto simple, texto enriquecido, imagen, fecha y checkbox.

A su vez el usuario puede configurar otras opciones del campo, por ejemplo si será requerido o si acepta múltiples valores.

Una vez que termine de configurar el campo y haga clic en el botón agregar el mismo se incorporará a una grilla de campos asociados al formulario:

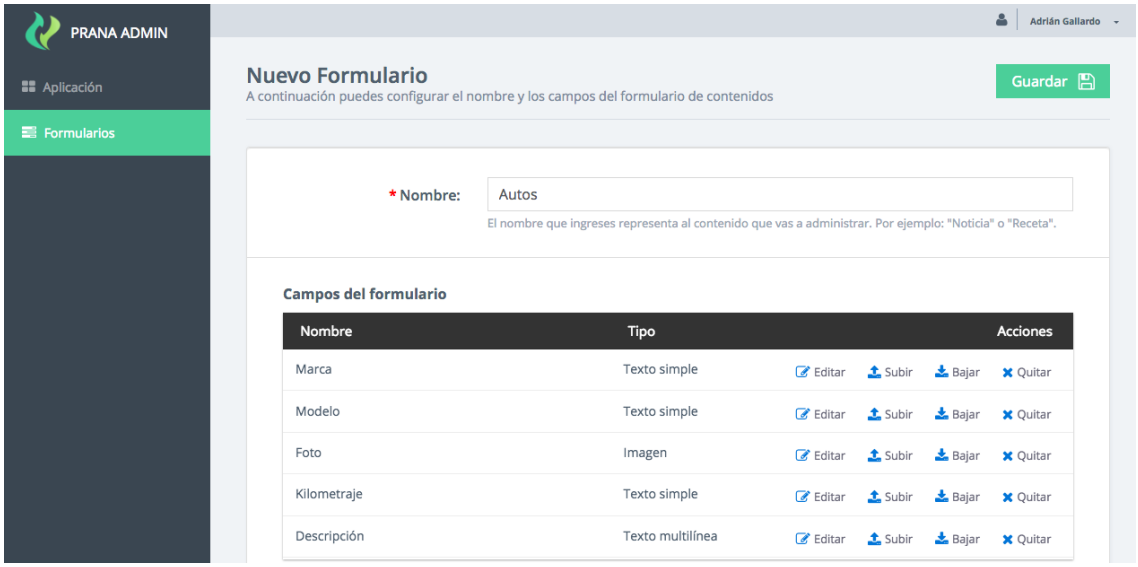
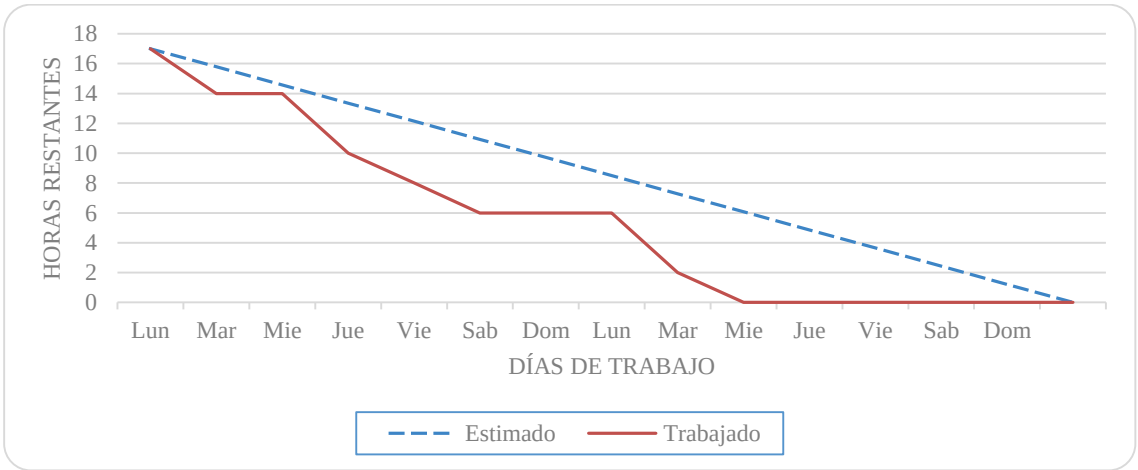



*Figura 31: Formulario con campos asociados (rol usuario-cms).*
*Fuente: Elaboración propia.*

En esta grilla de campos cada registro tendrá botones que permitirán editar o quitar el mismo. Además se podrá configurar el orden de aparición del mismo mediante los botones subir o bajar.

### 20.5.2. Diagrama Burndown Sprint 2




## 20.6. Sprint 3

En este Sprint se trabajó en la creación del formulario dinámico para la carga de contenidos y su posterior guardado. Dicho formulario se genera teniendo en cuenta los atributos (campos) configurados para las entidades (formularios).

Tabla 10
*Sprint Backlog del Sprint 3*

| Historia / SubTareas | Estimación original | Estimación actualizada | Sobreestimado Subestimado | Tiempo Trabajado |
|---|---|---|---|---|
| **CMS_006: Administrar contenido por entidad** | | | | |
| Menú de acceso a contenidos por entidad | 1 | 1 | 0 | 1 |
| Listado de contenidos por entidad | 6 | 4 | 2 | 4 |
| Definir arquitectura para generación de formulario dinámico | 4 | 10 | -6 | 10 |
| Armado dinámico del formulario de contenidos | 8 | 8 | 0 | 8 |
| Componente tipo dato texto | 1 | 1 | 0 | 1 |
| Componente tipo dato enriquecido | 3 | 3 | | |
| Componente tipo dato fecha | 2 | 3 | -1 | 3 |
| Componente tipo dato opción | 1 | 1 | 0 | 1 |
| Componente tipo dato imagen | 4 | 5 | -1 | 5 |
| Carga de campos con opción múltiple | 4 | 3 | 1 | 3 |
| Validación de datos del componente | 4 | 4 | 0 | 4 |
| Guardado de datos del formulario de contenidos | 4 | 4 | 0 | 4 |
| **TOTAL** | 42 | 47 | -5 | 44 |

### 20.6.1. Resultados del Sprint 3

A continuación se presentan las interfaces desarrolladas tras completar la tercera iteración.

Cuando el usuario da de alta uno o más formularios en el menú comienza a mostrare un ítem con el nombre “Contenidos” que al hacer clic despliega un submenú donde se listan los nombres de dichos formularios. Al hacer clic se redirige al usuario al listado de contenidos de dicho formulario.

La primera vez que el usuario ingresa en esta pantalla lógicamente no posee contenidos cargados, por tanto se muestra un mensaje aclaratorio. Se muestra además un botón que dirige al formulario de carga del contenido.

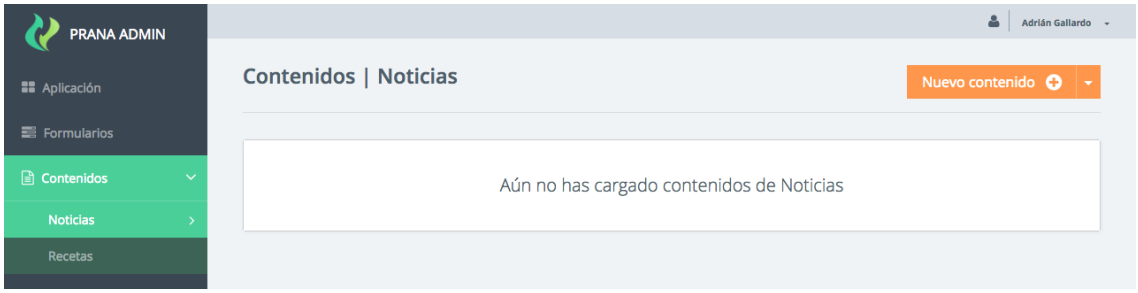



*Figura 32: Sección contenidos del formulario "Noticias". Sin contenido cargado. (rol usuario-cms). Fuente: Elaboración propia.*

A continuación se muestra un ejemplo de un posible formulario configurado por el usuario para administrar noticias.

Se recuerda que los campos son configurados por el usuario cuando da de alta o edita un formulario. Por cada campo puede definir el nombre del mismo, si será requerido, el tipo de dato que permite cargar, el orden, si se muestra en la grilla, y si acepta múltiples valores (en este caso, permite configurar un mínimo y máximo de repeticiones)

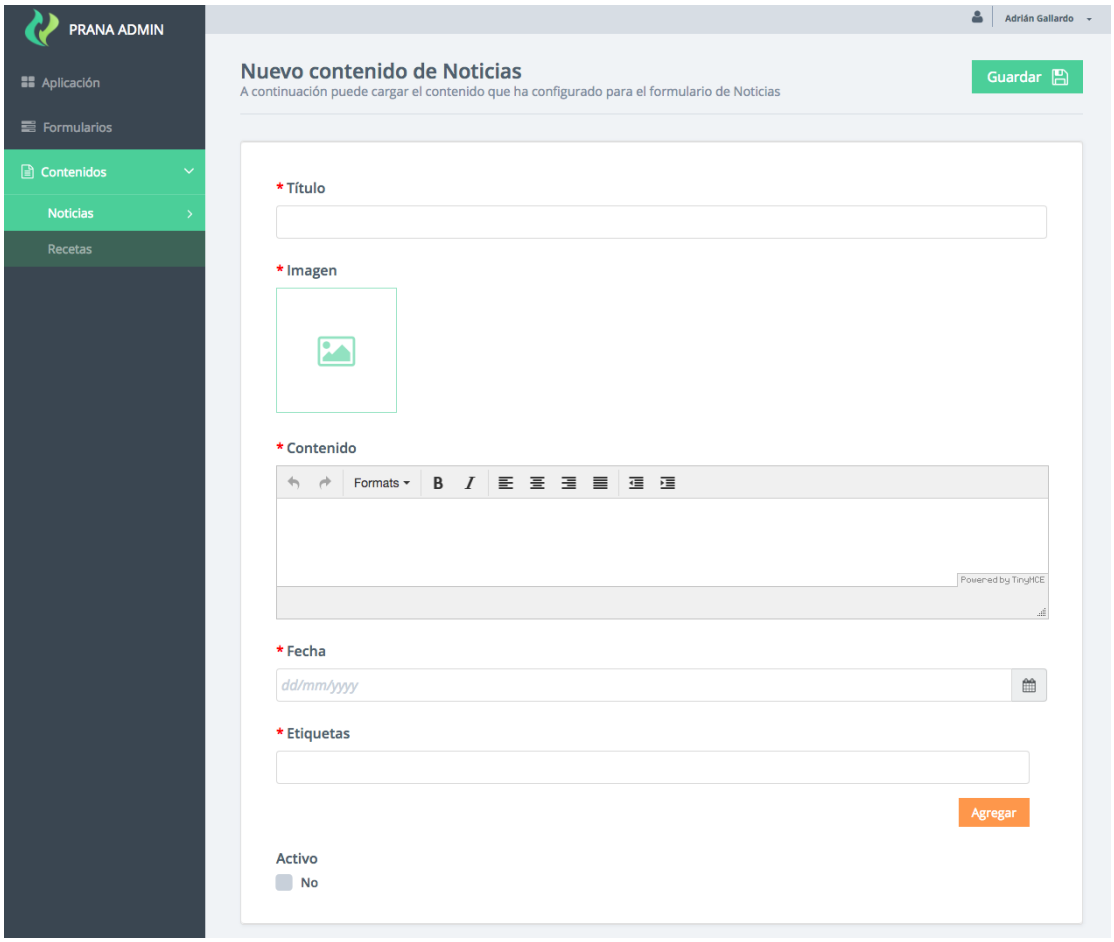



*Figura 33: Ejemplo de formulario de contenido generado automáticamente. Sin contenido cargado (rol usuario-cms). Fuente: Elaboración propia.*

Como puede observarse este formulario permite la carga de 6 campos:

- Título, de tipo texto;
- Imagen, de tipo imagen;
- Contenido, de tipo texto enriquecido;
- Fecha, de tipo fecha;
- Etiquetas, de tipo texto, pero acepta múltiples entradas, mínimo 1 y máximo 3;
- Activo, de tipo checkbox u opción simple.

Los campos que el usuario definió como requeridos para este formulario se marcan con un asterisco de color rojo.

A continuación se muestra cómo se visualiza este formulario luego de que el usuario haya cargado todos sus campos.

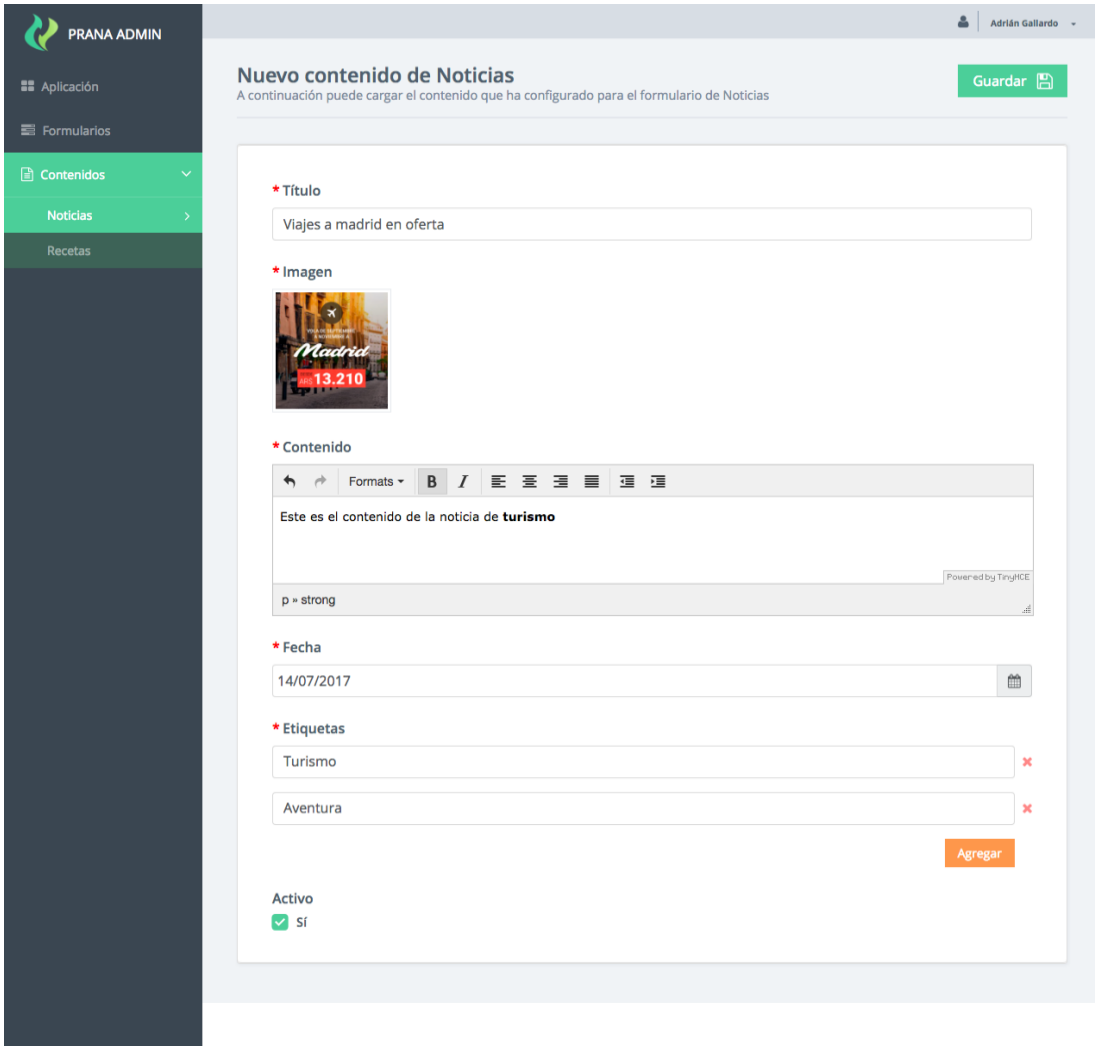



*Figura 34: Ejemplo de formulario de contenido con los datos cargados por el usuario (rol usuario-cms). Fuente: Elaboración propia.*

Una vez que el usuario guarda los datos, si no hay errores, se redirige se notifica con un mensaje de éxito y se redirige al listado de contenidos. Ahora el usuario puede visualizar en la grilla el contenido cargado.

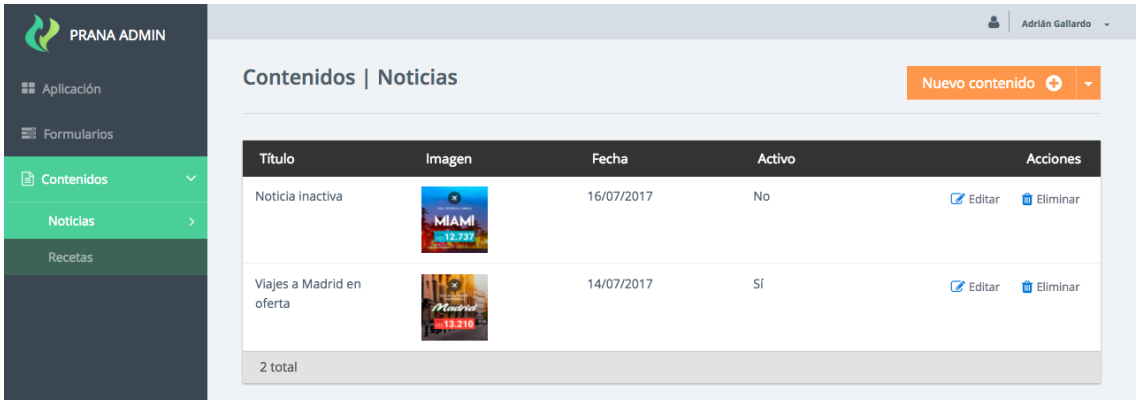
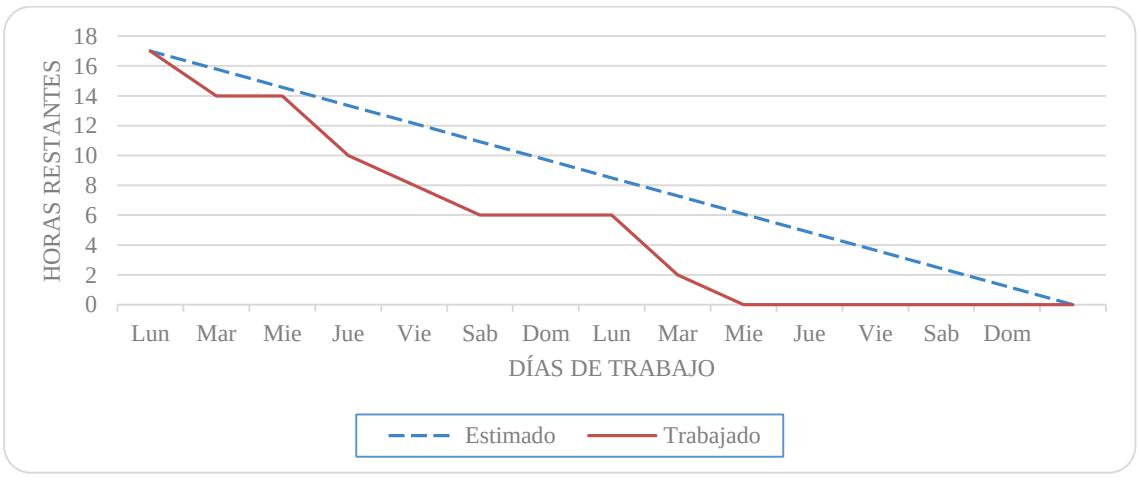



*Figura 35: Pantalla de listado de contenidos cargados para un formulario (rol usuario-cms). Fuente: Elaboración propia.*

En esta grilla también puede verse cómo se tienen en cuenta los tipos de datos de los campos. Por ejemplo los campos de tipos de imagen muestran una miniatura de la misma, y los de tipo checkbox muestran el texto “Sí” o “No” de acuerdo a si se activó o no la opción.

Por último, se recuerda que en esta grilla se muestran sólo los atributos que el usuario haya marcado con la opción de “Mostrar en grilla de contenidos” y en el orden configurado. Si el usuario no marcó ningún campo entonces se mostrarán los campos configurados como requeridos. En caso contrario se mostrarán hasta los primeros 5 campos del formulario.

### 20.6.2. Diagrama Burndown Sprint 3



## 20.7. Sprint 4

En este Sprint el objetivo fue desarrollar la aplicación cliente que utilizará el usuario para conectarse a su aplicación en PranaCMS.

Una vez conectado se pueden obtener los contenidos que haya cargado para alguno de los formularios que haya configurado.

A su vez se planificó el desarrollo de una aplicación de ejemplo para poder corroborar el funcionamiento y usabilidad de la API.

Tabla 11
*Sprint Backlog del Sprint 4*

| Historia / SubTareas | Estimación original | Estimación actualizada | Sobreestimado Subestimado | Tiempo Trabajado |
|---|---|---|---|---|
| **LIBRERIA_001: Conectar librería con aplicación en el CMS** | | | | |
| Definir interfaz de clase y desarrollar librería que se conecte con aplicación del cliente | 5 | 6 | -1 | 6 |
| Definir y desarrollar mecanismo de autenticación de la librería cliente con la aplicación CMS | 2 | 3 | -1 | 3 |
| Desarrollar aplicación demo para probar la funcionalidad de conexión con aplicación en CMS | 2 | 2 | | |
| **LIBRERIA_002: Mostrar lista de contenidos** | | | | |
| Obtener contenidos de una entidad y mostrar los datos en el template definido por el usuario | 12 | 10 | 2 | 10 |
| Filtrar contenidos de una entidad | 4 | 3 | 1 | 3 |
| Desarrollar aplicación demo para probar la funcionalidad de obtener y presentar el listado | 4 | 4 | | |
| **LIBRERIA_003: Mostrar detalle de contenido** | | | | |
| Obtener detalle del contenido y mostrar los datos en template definido por el usuario | 4 | 4 | 0 | 4 |
| Permitir interacción de tipo master-detail entre el listado de contenidos y la vista de detalles | 2 | 2 | 0 | 2 |
| Desarrollar applicación demo para probar funcionalidad de obtener y mostrar detalle | 2 | 2 | 0 | 2 |
| **TOTAL** | 37 | 36 | 1 | 30 |

### 20.7.1. Autenticación de la aplicación cliente con JWT

Para poder conectar el sitio web del usuario con su aplicación en PranaCMS es necesario contar con algún mecanismo de autenticación con el propósito de:

- Validar que la aplicación a la que se quiere conectar esté en estado activo;
- Validar que el usuario que se conecta con la aplicación envíe las credenciales pertinentes;
- Identificar con cuál de las aplicaciones de un usuario se está tratando de conectar (recordar que un usuario puede administrar más de una aplicación).

Con este propósito es que se implementará nuevamente un protocolo de autenticación por medio de tokens utilizando la libraría JWT, sólo que a diferencia de un Login común en el que se envía usuario y contraseña en este caso se enviaría el identificador y clave de la aplicación.

Con estos dos datos es posible identificar unívocamente la aplicación a la que se desea conectar. En respuesta, si los datos son válidos y la aplicación está activa, el servicio devolverá un token que deberá ser utilizado en las futuras peticiones para obtener los contenidos.

### 20.7.2. Obtención de contenidos

Para poder obtener los contenidos el usuario debe proveer mínimamente dos datos:

- El identificador de la entidad / formulario del cual se desean obtener los datos,
- El token descripto en el punto anterior.

Dicho token es importante ya que en su interior contiene información de la aplicación a la que pertenece por lo que es necesario para saber sobre qué entidades trabajar.

Adicionalmente el usuario podrá proveer un dato adicional para filtrar los resultados almacenados.

### 20.7.3. Referencia técnica

A continuación se hace una breve referencia técnica a algunos conceptos utilizados que es importante conocer para poder comprender mejor la solución desarrollada.

#### 20.7.3.1. TypeScript

TypeScript es una extensión de JavaScript que añade funcionalidades muy útiles como sintaxis de clase, interfaces, herencia y definición de tipo de datos (tanto para las propiedades, argumentos y resultados de los métodos).

Cualquier código válido en JavaScript es código válido en TypeScript, pero no así al revés, por lo que es necesario pre-compilar el código resultante antes de poder utilizarlo en la web.

#### 20.7.3.2. Promesas

Una promesa es un objeto que representa la espera de un resultado futuro. Este tipo de objeto es muy utilizado en JavaScript cuando se realizan procesos asíncronos, es decir, que requieren de un cierto tiempo (indeterminado) hasta obtener el resultado solicitado. Un ejemplo típico de proceso asíncrono es cuando se realiza una petición al servidor para obtener una respuesta.

Un objeto promesa luego puede resolverse (método "resolve") o fallar (método "reject"). Si se resuelve obtenemos como argumento el valor esperado y se ejecuta la función que hayamos definido en caso de éxito. Si se rechaza obtenemos como argumento un valor que identifica el error ocurrido y se ejecuta la función que hayamos configurado para manejar estos problemas.

A continuación se presenta un resumen del funcionamiento interno de una promesa mediante un diagrama BPMN:

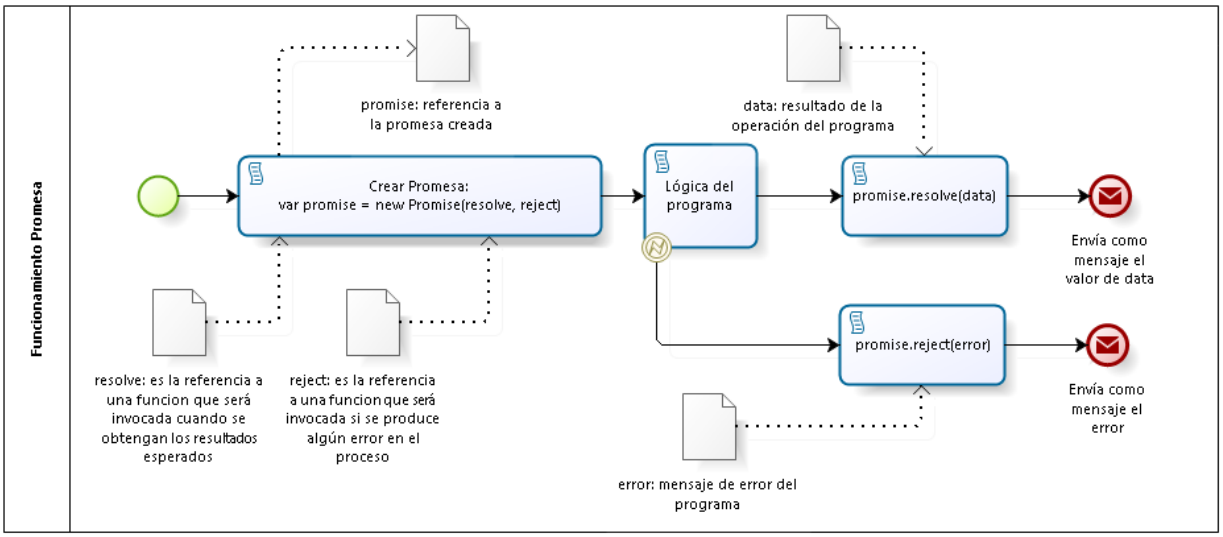



*Figura 36: Diagrama BPMN que grafica el funcionamiento de una promesa.*
*Fuente: Elaboración propia.*

#### 20.7.3.3. Web Components

Un Web Component (o componente web) es un concepto muy utilizado actualmente por muchos de los frameworks más populares de JavaScript como Angular, React, Polymer, Aurelia, Ember, Riot o Vue.

Un componente web está formado por una vista definida en HTML y su lógica en formato JavaScript. Opcionalmente también pueden tener asociadas reglas de estilo CSS donde se define la apariencia del componente (colores, layout, entre otros atributos).

Cada componente web es representado mediante una etiqueta HTML personalizada, por ejemplo "<mi-componente>". De esta manera cada vez que se quiera incluir una instancia del componente en una página web podemos hacerlo de manera simple como si se tratara de cualquier otra etiqueta HTML.

Un componente web puede recibir parámetros de entrada y comunicarse con otros componentes (la forma de comunicación depende del framework utilizado).

### 20.7.4. Resultados del Sprint 4

La librería cliente se compone de una clase en TypeScript y de un componente web que hace uso de la librería Riot.js.

#### 20.7.4.1. Clase Prana

A continuación se presenta la interfaz principal de la clase Prana (sin ahondar en detalles de implementación) y posteriormente se explica su funcionamiento:

```
interface IPrana {

  static api_promise:Promise<string>

  static connect(token:string,
  secret:string):Promise<string>

  static fetchResults(entity:string,
  query?:any):Promise<any[]>

}
```

El método "connect" es el encargado de realizar la conexión con la aplicación en el CMS. Para esto recibe como parámetros el "identificador" y "clave" de la aplicación (en la librería llamados "token" y "secret" por ser los nombres en inglés que se utilizan comúnmente en este tipo de librerías) y devuelve una promesa (objeto "Promise") que de ser exitosa resolverá con el token de la aplicación generado. En caso de ser rechazada (por ejemplo por datos incorrectos o un timeout del servidor) se recibirá un mensaje del error ocurrido.

Dicha promesa será almacenada en la propiedad "api_promise" listada con anterioridad en la clase Prana.

El método “fetchResults” provee la API necesaria para poder obtener los contenidos de una entidad. Para esto requiere el identificador de la entidad (parámetro “entity”) y opcionalmente un parámetro “query” para refinar la búsqueda. La librería automáticamente agregará en la cabecera de autenticación HTTP el token previamente obtenido en el paso anterior. El resultado de este método será una promesa que, una vez resuelta, devolverá un arreglo de los contenidos encontrados.

Puede suponerse que el método “fetchResults” no debe permitir ser invocado hasta que la aplicación no se haya autenticado y recibido el token correspondiente. No obstante gracias al uso de promesas es posible invocar este método antes, durante o después de haberse autenticado ya que internamente este método esperará a que la promesa de autenticación almacenada en la propiedad “api_promise” se resuelva con un token válido.

A continuación se ilustra mejor esta secuencia de pasos mediante un diagrama BPMN para facilitar su entendimiento.

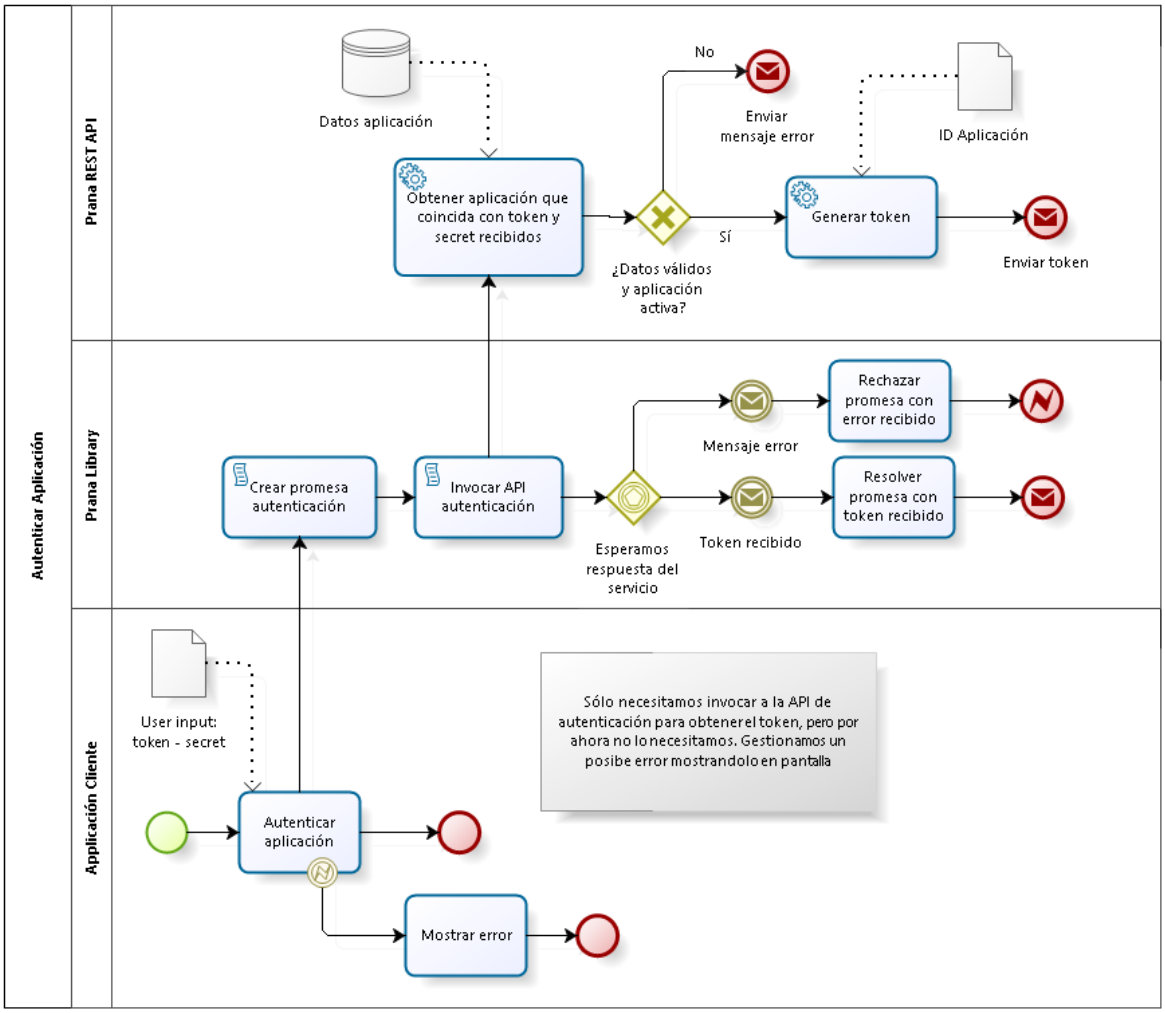



*Figura 37: Diagrama BPMN que grafica el proceso de autenticación con la aplicación mediante el uso de promesas. Fuente: Elaboración propia.*

El diagrama anterior nos muestra lo que implica la acción de autenticación para la aplicación cliente: sólo debe llamar al método de autenticación pasando las credenciales correspondientes y en caso de ocurrir un error mostrarlo.

En este punto a la aplicación no le interesa realizar ningún paso adicional ya que el token lo utilizará recién cuando invoque a la API de obtener resultados tal como se muestra en el siguiente proceso.

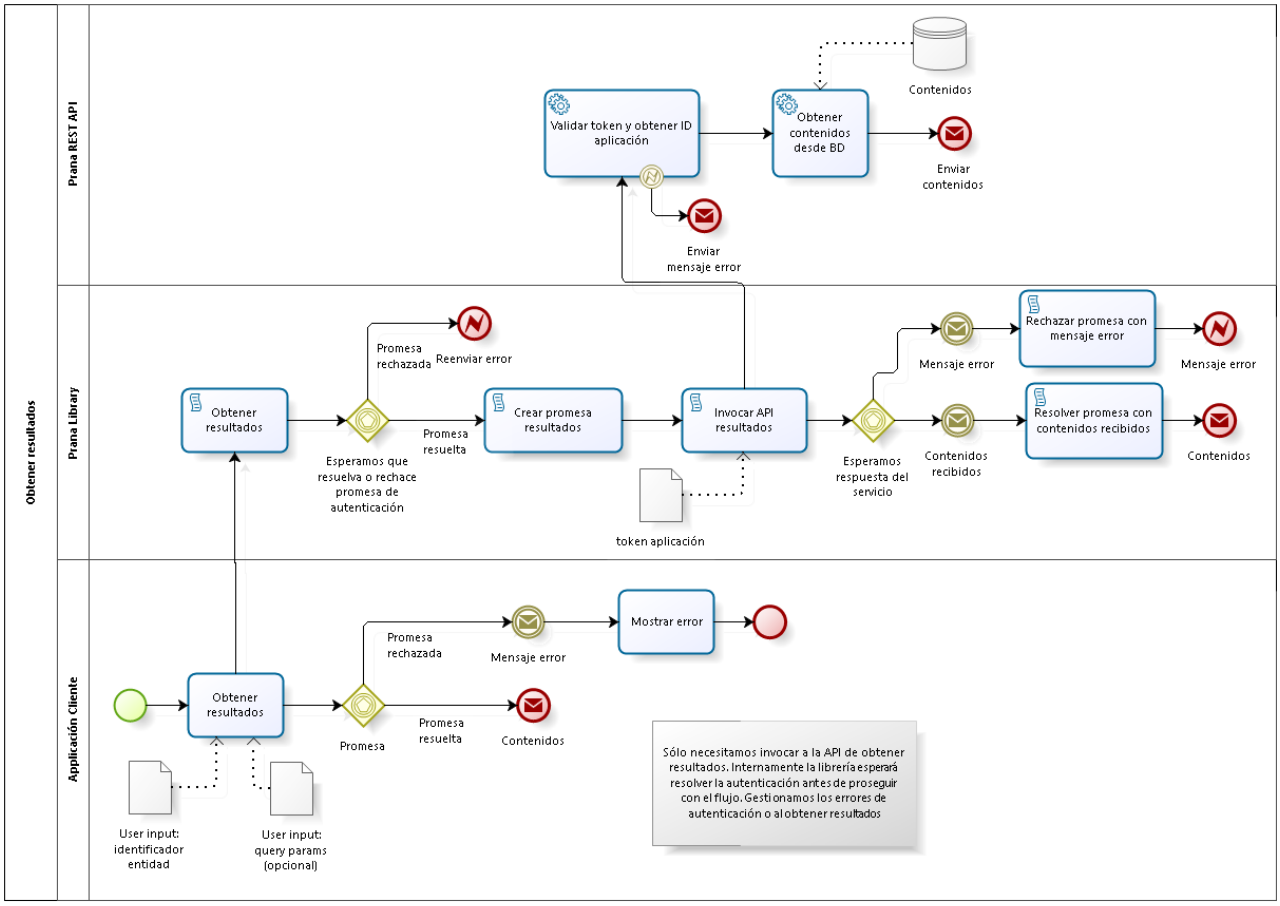



*Figura 37: Diagrama BPMN que grafica el proceso de obtención de resultados mediante el uso de promesas. Fuente: Elaboración propia.*

En este diagrama vemos que el proceso de obtener resultados es también muy simple: la aplicación cliente sólo debe invocar la API de obtener resultados pasando como datos el identificador de la entidad y una query de filtrado (opcional) y se obtiene como respuesta una promesa que resolverá con los contenidos buscados.

Mediante esta arquitectura resolvemos internamente el proceso de validación de la aplicación y obtención del token. Esto le provee al usuario de la librería de los siguientes beneficios:

- No debe preocuparse acerca del orden de ejecución de las llamadas de autenticación y resultados;

- Ante cualquiera de los 2 métodos sólo puede obtener 2 resultados posibles: un mensaje de éxito con los datos correspondiente o un mensaje de error;
- El método para obtener resultados se encarga de gestionar la autenticación de la aplicación y de enviar el token en la llamada al servicio web de forma transparente para el usuario.

#### 20.7.4.2. Componente Web <prana-content>

Con el objetivo de simplificar y abstraer toda la complejidad de un código de programación al usuario de la librería es que se creó un componente web.

Mediante dicho componente el usuario podrá obtener los contenidos que esté gestionando en PranaCMS.

Ejemplo de uso:

```
<prana-content from=”noticias” find=”destacada=true”>
    <div class=”noticia”>
          <h1>Título: { item.titulo }</h1>
          <p>Contenido: { item.contenido }</p>
    </div>
</prana-content>
```

*Figura 38: Ejemplo del código HTML a utilizar para integrar contenidos.*
*Fuente: Elaboración propia.*

En negrita se destacan las parte principales que se explicarán a continuación:

- **<prana-content></prana-content>:** son las etiquetas que permiten generar automáticamente la instancia del componente web que se encargará de obtener los resultados desde el servidor.
  Todo lo que se encuentra dentro de estas etiquetas se utilizará como template para generar la vista resultante repitiendo el mismo una vez por cada resultado obtenido.

- **from=”…”:** en este atributo el usuario deberá ingresar el identificador de la entidad administrada. En este ejemplo es la entidad o formulario “noticias”.

- **find=”…”:** mediante este atributo se podrá realizar un filtrado de los contenidos esperados. En este ejemplo se obtendrán solo las noticias que tengan un campo con el identificador “destacado” y cuyo valor sea “true”. Este atributo es opcional.

- **{ ítem.identificador_campo }:** Como ya se anticipó el template se utilizará una vez por cada contenido encontrado. En cada iteración “ítem”

representará el valor del contenido actual. Entonces mediante la expresión {item.identificador_campo} lo que estamos indicando es que se muestre en pantalla el valor de dicho campo del ítem actual. En el ejemplo se mostrarían los valores de los campos “titulo” y “contenido” de cada una de las noticias obtenidas.

#### 20.7.4.3. Ejemplo de uso completo

A continuación se presenta como sería el código fuente completo que debe implementar el usuario para poder mostrar los contenidos de su entidad noticias:

```
<html>
    <head>
        <title>Noticias destacadas</title>
    </head>
    <body>

        <!-- Mostramos las noticias destacadas -->
        <prana-content from="noticias" find="destacada=true">
            <div class="noticia">
                <h1>Título: { item.titulo }</h1>
                <p>Contenido> { item.contenido }</p>
            </div>
        </prana-content>

        <!-- Incluimos la libreria prana.js -->
        <script src="prana.js"></script>

        <!-- Configuramos la conexion con PranaCMS -->
        <script>
            var token = "NDk5NDQuMTQ5NjcwNDEyODI3OA";
            var secret = "gclocfpgirq401ac211mrt3xr1hkpxwr6v7q";

            Prana.connect(token, secret);
        </script>
    </body>
</html>
```

*Figura 39: Ejemplo completo del código HTML a utilizar para integrar contenidos mediante el uso de la librería cliente. Fuente: Elaboración propia.*

## 20.8. Sprint 5

Este es el último Sprint planificado. En el mismo se desarrollarán las funcionalidades que le permitirán al usuario administrador ingresar para poder gestionar la información de las aplicaciones creadas y los usuarios.

Tabla 12
*Sprint Backlog del Sprint 5*

| Historia / SubTareas | Estimación original | Estimación actualizada | Sobreestimado Subestimado | Tiempo Trabajado |
|---|---|---|---|---|
| **BACKEND_001: Login administrador** | | | | |
| Restringir acceso a usuarios de tipo administrador | 2 | 2 | 0 | 2 |
| Menú administrador | 1 | 1 | 0 | 1 |
| **BACKEND_002: Administrar cuentas de usuario** | | | | |
| Listar usuarios creados en el sistema | 4 | 4 | 0 | 4 |
| Edición de datos de usuario | 1 | 1 | 0 | 1 |
| Crear nuevo usuario | 2 | 2 | | |
| Activar o desactivar cuenta de usuario | 1 | 1 | | |
| **BACKEND_003: Administrar aplicaciones** | | | | |
| Listar aplicaciones creadas en el sistema | 4 | 4 | 0 | 4 |
| Edición de datos de aplicación | 1 | 1 | 0 | 1 |
| Activar o desactivar aplicación | 1 | 1 | 0 | 1 |
| **TOTAL** | 17 | 17 | 0 | 14 |

### 20.8.1. Resultados del Sprint 5

A continuación se presentan las interfaces desarrolladas tras completar la última iteración.

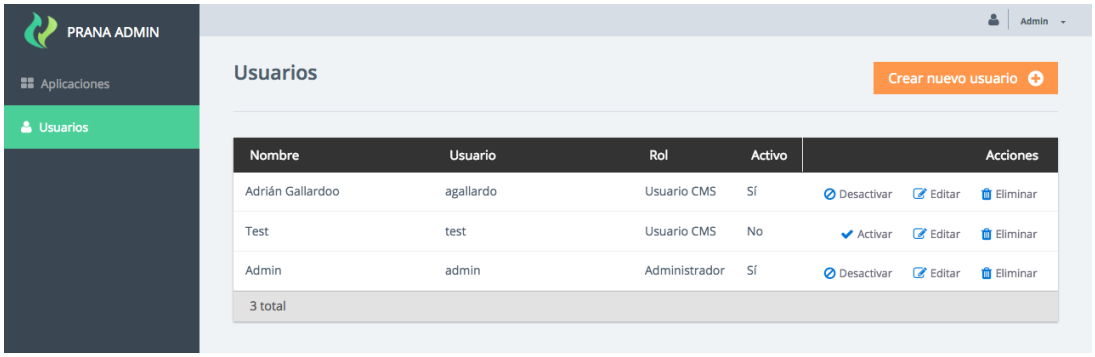



*Figura 39: Pantalla de listado de usuarios (rol administrador).*
*Fuente: Elaboración propia.*

En esta pantalla se muestra una lista de los usuarios de la aplicación, tanto los activos como los inactivos.

Mediante una acción rápida en la grilla podemos cambiar el estado activo / desactivo del usuario. Además podemos eliminarlo (esta acción lo removerá de forma permanente) o bien editarlo.

Vemos en esta pantalla que además presenta un botón para crear usuario. Por el momento este será el único medio disponible. A futuro se desarrollará una web para promocionar Prana CMS y se permitirá el registro de nuevos usuarios desde allí.

La pantalla de edición nos permite, además ingresar los datos generales, ingresar un nombre de usuario (que deberá ser único), reiniciar el password y definir el rol del mismo (actualmente Administrador o Usuario CMS). Además se muestra (en formato sólo lectura) la fecha en la que fue creado.

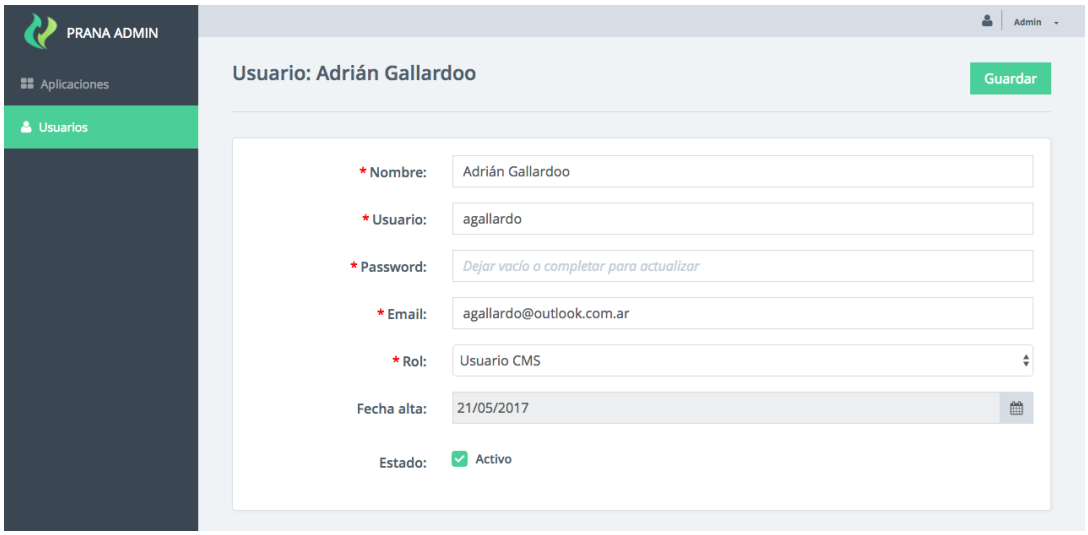



*Figura 40: Formulario de edición de un usuario del sistema (rol administrador). Fuente: Elaboración propia.*

Por último se muestran las pantallas de listar y editar aplicaciones. En este caso no existe una funcionalidad para dar de alta ya que sólo pueden hacerlo los usuarios con el rol Usuario CMS cuando acceden con su cuenta.


*Figura 41: Pantalla de listado de aplicaciones (rol administrador). Fuente: Elaboración propia.*

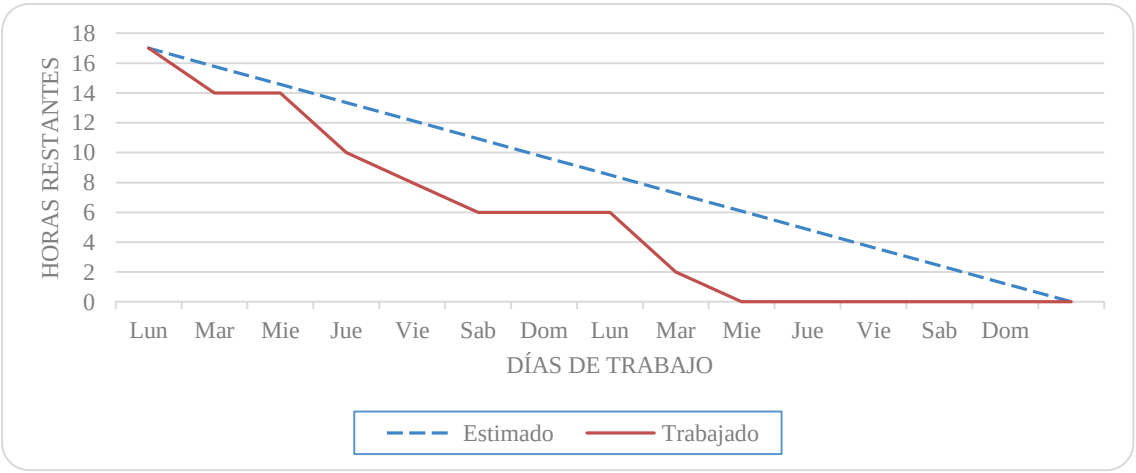
Vemos que también podemos activar o desactivar una aplicación desde esta grilla. Cuando una aplicación está inactiva el usuario no podrá seleccionarla para editar

los formularios y contenidos, no obstante cualquier web que esté integrada mediante la Librería Cliente seguirá funcionando normalmente.


*Figura 42: Pantalla de edición de los datos de una aplicación (rol administrador). Fuente: Elaboración propia.*

El formulario de edición es muy similar al que ve un usuario común, sólo que permitimos desde aquí cambiar su estado y la opción de regenerar la clave no está disponible.

### 20.8.2. Diagrama Burndown Sprint 5




Vemos que este sprint se terminó cómodamente antes de lo previsto ya que la carga horaria estimada era mucho menor producto de la baja complejidad de las historias y por tratarse de funcionalidades muy similares a las ya desarrolladas en otras historias anteriores.

# 21.Modelo de negocio

El sistema de CMS desarrollado durante este proyecto es viable de constituirse como una solución de software por la que algunos clientes estén dispuestos a invertir dinero para poder beneficiarse con las virtudes que ofrece.

Es por esta razón que resulta conveniente realizar el planteo de algún modelo de negocio, aunque sea preliminar y pueda cambiar a futuro, para ir definiendo y planificando su posterior desarrollo y crecimiento.

Una de las herramientas de modelado de negocio más difundida y utilizada por los emprendedores hoy en día es el Modelo Canvas.

Para constituir este modelo es necesario realizar un análisis en cada uno de los 9 bloques del sistema. A continuación se listan dichos bloques y luego se profundizará en su desarrollo dando una breve introducción de su significado y cómo se abordará el mismo en relación a PranaCMS:

- Segmentos del mercado
- Propuesta de valor
- Canales
- Relación con el cliente
- Fuentes de ingresos
- Recursos claves
- Actividad clientes
- Asociaciones claves
- Estructura de costos

## 21.1. Segmentos del mercado

Esta sección analiza y define los diferentes grupos a los que el producto será dirigido. Un modelo de negocio podrá definir uno o varios segmentos.

El segmento de mercado principal al que apunta PranaCMS es a profesionales independientes en las áreas de Diseño Gráfico y Desarrollo Web, o bien que trabajan en alguna pequeña o mediana empresa dedicada a dichos rubros.

También se considera la posibilidad de que profesionales idóneos en el rubro de desarrollo de sistemas puedan integrar PranaCMS como una alternativa para permitir la gestión de contenidos en sus desarrollos y reducir así parte de sus costos y esfuerzos.

Como requisito mínimo, el perfil de clientes al que va dirigida la solución debe poseer conocimientos básicos que le permiten generar un sitio web en formato HTML.

## 21.2. Propuesta de valor

Se describirán los atributos del producto que crean valor para un segmento de mercado específico, indicando qué problema o necesidad satisface.

Lo que se propone como valor para el cliente es brindar una solución simple que le permita encarar de forma rápida y económica el desarrollo de un sitio web con contenido administrable.

Esta tarea posee cierta complejidad técnica y tecnológica inherente, ya que involucra por un lado el manejo de entidades y bases de datos y también requiere conocimientos de lenguajes de programación para generar contenido en forma dinámica.

Además de dicha complejidad implementar sitios con contenido autoadministrable suele resultar ser una tarea tediosa, repetitiva y que consume mucho tiempo y esfuerzo incluso por parte de desarrolladores experimentados.

Entonces, los puntos fuertes de la propuesta de valor son:

- Posibilitar el desarrollo de sitios web con contenido autoadministrable a profesionales que carecen del conocimiento técnico sobre servidores, bases de datos y lenguajes de programación;
  El beneficio para este tipo de clientes es que podrán ofrecer este tipo de soluciones a sus propios clientes, algo que de otra manera no podrían hacer y tendrían que terciarizar parte o la totalidad de su trabajo.
- Permitir integrar el contenido auto administrado por PranaCMS en cualquier sitio web, nuevo o ya existente, sin importar el lenguaje de programación o plataforma de servidor utilizados.
  El beneficio para este tipo de clientes es poder integrar una solución flexible, económica y rápida para que sus propios clientes administren el contenido de su web reduciendo así costos de desarrollo, lo que les permitirá obtener más ganancias u ofrecer precios más competitivos.

## 21.3. Canales

Los canales hacen referencia a dos puntos del Marketing Mix: plaza y promoción. Por plaza se entiende al medio físico o digital a través del cual se realizará el intercambio comercial (producto o servicio a cambio de algún rédito, por lo general económico). Por promoción se refiere al modo o medio utilizado para dar a conocer el producto o servicio entre sus potenciales clientes.

Por lo general, este tipo de soluciones suelen promocionarse y contratarse a través de la web. Entonces, resulta indispensable el desarrollo de una página web en la que se enuncien las virtudes de PranaCMS y se ofrezca la posibilidad de generar una cuenta y suscribirse a algún plan.

Además se promocionará la herramienta de manera personal, tanto en workshops a los que asisten profesionales del Diseño Gráfico y Web como también concurriendo y promocionando directamente en pequeñas y medianas empresas del rubro.

Se ofrecerán cursos de capacitación gratuitos orientados a profesionales que se estén insertando en el desarrollo web. Dichas capacitaciones se pueden dar tanto de manera presencial como también de manera virtual a través de videos multimedia. En las mismas se enseñarían conceptos básicos para el desarrollo de páginas web (como HTML, CSS o JavaScript) y se presentará a PranaCMS como alternativa simple y económica para incluir contenido autoadministrable en la web.

Por último, se trabajará en generar presencia y conocimiento de la herramienta en foros dedicado al diseño y desarrollo de páginas web.

### 21.4. Relación con cliente

Describe la relación que establece la empresa con el cliente luego de realizada la compra o uso del servicio. Esta relación busca generar vínculos que le permitan por un lado a la empresa estar al tanto de las necesidades o intereses del cliente, como también así estar al tanto de sus problemas para resolver cualquier inquietud.

Para lograr una buena relación con el cliente es indispensable disponibilizar una fuente confiable y completa de documentación.

Dicha documentación estará presente dentro de la misma web (accesible tanto para los usuarios registrados como para los usuarios anónimos), a través de videos gratuitos en YouTube y en plataformas multimedia educativas (como por ejemplo Udemy) y brindando soporte activo en foros de desarrolladores (como por ejemplo StackOverflow).

Además, se le pedirá al cliente que conteste encuestas que permitan conocer el grado de satisfacción del mismo respecto de los servicios recibidos, y además para permitirle realizar sugerencias sobre futuras funcionalidades que desearían que la herramienta de CMS les brinde.

### 21.5. Actividades clave

En este módulo se describen las actividades más importantes vinculadas al producto o servicio que se ofrecerá y a la propuesta de valor ofrecida. Sin estas actividades el desarrollo o mantenimiento del producto o servicio no se podría llevar adelante.

La actividad principal consiste en el desarrollo y mantenimiento del CMS. También se dedicará tiempo y esfuerzo constante para relevar nuevas necesidades en los

clientes que orienten el desarrollo de nuevas funcionalidades que extiendan las características del producto ofrecido.

Además, resulta importante realizar tareas de capacitación con potenciales nuevos clientes y brindar soporte a los clientes actuales, todo esto con el objetivo de expandir y afianzar el mercado respectivamente.

Por último, realizar promoción constante para captar nuevos clientes ya sea en forma personal como también mediante plataformas digitales.

## 21.6. Recursos claves

Son los activos con los que se ha de contar para llevar a cabo el modelo de negocio.

En el caso de nuestro producto se requiere de los siguientes recursos:

- **Físicos:**
  - Infraestructura: servidores y bases de datos en donde desplegar el sistema.
  - Computadoras para las actividades de diseño y desarrollo.
  - Software open source y propietario.

- **Recursos Humanos**:
  - Project Manager: estará a cargo de la coordinación del equipo, la gestión del mantenimiento del producto y el desarrollo de nuevas funcionalidades;
  - Ingeniero en software: El ingeniero en software estará a cargo de las fases de análisis y diseño del producto. Será el responsable técnico del equipo.
  - Programadores: Fundamentales para el desarrollo y mantenimiento del producto.
  - Diseñadores Gráficos: generarán la identidad visual de la empresa y diseñarán tanto las interfaces gráficas como también cualquier soporte informativo o promocional que se requiera.
  - Testers: Fundamentales para probar el correcto funcionamiento del sistema en todas sus etapas.
  - Docentes/capacitadores: Establecer contacto directo y continuo con los clientes reales y desarrollar actividades para atraer y persuadir a clientes potenciales.

- **Financieros:** necesarios para adquirir y contratar los recursos anteriormente citados.

## 21.7. Alianzas claves

Las alianzas claves tienen que ver con la red de proveedores y socios que contribuyen al funcionamiento del modelo de negocio.

Para llevar adelante el proyecto resulta indispensable conseguir una buena infraestructura de servidores en donde desplegar la solución. Que sea confiable, escalable y económica. Una alternativa viable sería la contratación de servicios a medida en la plataforma Google Cloud.

También para las etapas tempranas de promoción resulta importante contar con el apoyo de instituciones académicas en donde se forman profesionales del Diseño Gráfico y desarrollo de Sistemas, como por ejemplo instituciones universitarias, que den apoyo y permitan la realización de charlas, workshops, cursos y congresos para la promoción del producto.

## 21.8. Estructura de costos

La estructura de costos está compuesta por todos aquellos gastos en los que se debe incurrir para poder llevar adelante el plan de negocio. Está directamente relacionado con los otros bloques analizados, como por ejemplo con los recursos y actividades clave, promoción o desarrollo de la propuesta de valor.

Algunos costos que se encuentran son los siguientes:

- Pago de honorarios a desarrolladores y diseñadores;
- Alquiler de un espacio de trabajo;
- Pago de servicios, como internet o electricidad;
- Pago de licencias de software para el diseño y desarrollo;
- Pago de publicidad y promoción en medios digitales;
- Pago por el servicio de infraestructura de servidores.

## 21.9. Fuentes de ingresos

Existen muchos productos CMS gratuitos y muy completos en el mercado, por lo que cobrar por el uso del producto puede resultar difícil hasta no hacer conocidas las bondades que se ofrecen.

Es por esta razón que se piensa en la implementación de uno o más de los siguientes patrones para generar ingresos sin depender exclusivamente de la venta de licencias:

- **Carnada y anzuelo:** este patrón se refiere a cuando se ofrece una muestra gratis y luego se cobra por seguir utilizando el producto. Un ejemplo de esto sería

permitir que el usuario cree sus primeras aplicaciones de manera gratuita y que luego deba pagar un costo por cada aplicación nueva. Este patrón es muy útil cuando el producto no es conocido pero se tiene mucha confianza en los beneficios que se derivan de su uso para los potenciales clientes.

- **Freemium:** consiste en disponibilizar algunas funciones del producto de manera gratuitas y otras de pago. En nuestro caso, podríamos por ejemplo permitir que el usuario cree de manera gratuita una determinada cantidad de entidades o de contenido por entidad, y luego cobrar un cargo si requieren excederse de ese límite. También pueden darse servicios adicionales a los clientes que pagaron una cuenta Premium, como por ejemplo habilitando un modo edición en su página web que les permita actualizar los contenidos directamente sin tener que acceder al CMS.

- **Suscripción:** consiste en cobrar un monto económico fijo por el uso del producto o servicio. Pueden definirse y crearse distintos planes que respondan a distintas necesidades del cliente. Relacionado con el caso anterior puede existir un plan gratis y otro pago, por ejemplo cuando se necesiten gestionar más de 100 contenidos por entidad.

- **Plataforma multilateral y comisión:** a futuro podría desarrollarse un Marketplace en donde desarrolladores puedan crear y ofrecer en venta distintos componentes o extensiones que resulten útiles para otros clientes. Ejemplo de este tipo podrían ser carritos de compra, integración con medios de pago, integración con servicios de correo electrónico o redes sociales, entre otros. Entonces cobraríamos una comisión por las ventas concretadas a través del Marketplace.

- **Capacitaciones:** muchas empresas de software libre consiguen sus principales ingresos capacitando o brindando soporte diferencial a sus clientes. Esto podría resultar una alternativa viable para cuando la herramienta madure y se encuentre bien posicionada en el mercado.

## 22.Flujo de fondos

Se presenta una tabla con un posible flujo de fondos estimado para poder llevar adelante el proyecto, contemplando los primeros 12 meses de vida del emprendimiento.

Tabla 12
*Flujo de fondos proyectados para el primer año del emprendimiento*

| | Preinicio | Mes 1 | Mes 2 | Mes 3 | Mes 4 | Mes 5 | Mes 6 | Mes 7 | Mes 8 | Mes 9 | Mes 10 | Mes 11 | Mes 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Capital inicial** | 100.000 | 920.000 | 806.000 | 692.000 | 578.000 | 459.625 | 344.438 | 232.563 | 126.313 | 41.313 | -6.188 | -21.188 | 1.313 |
| **ENTRADAS EFECTIVO** | | | | | | | | | | | | | |
| Ventas en efectivo | 0 | 0 | 0 | 0 | 2.500 | 6.250 | 12.500 | 25.000 | 50.000 | 100.000 | 150.000 | 200.000 | 250.000 |
| Inversiones capital | 900.000 | 0 | | | | | | | | | | | |
| **TOTAL COBROS** | 900.000 | 0 | 0 | 0 | 2.500 | 6.250 | 12.500 | 25.000 | 50.000 | 100.000 | 150.000 | 200.000 | 250.000 |
| **Total disponible** | 1.000.000 | 920.000 | 806.000 | 692.000 | 580.500 | 465.875 | 356.938 | 257.563 | 176.313 | 141.313 | 143.813 | 178.813 | 251.313 |
| **SALIDAS EFECTIVO** | | | | | | | | | | | | | |
| Compras - Notebooks | 80.000 | | | | | | | | | | | | |
| Pago suedos | | 100.000 | 100.000 | 100.000 | 100.000 | 100.000 | 100.000 | 100.000 | 100.000 | 100.000 | 100.000 | 100.000 | 100.000 |
| Alq. co-working + servicios | | 12.000 | 12.000 | 12.000 | 12.000 | 12.000 | 12.000 | 12.000 | 12.000 | 12.000 | 12.000 | 12.000 | 12.000 |
| Infraestructura GoogleCloud | | 1.500 | 1.500 | 1.500 | 3.000 | 3.000 | 5.000 | 5.000 | 5.000 | 10.000 | 10.000 | 15.000 | 20.000 |
| Publicidad | | 0 | 0 | | 5.000 | 5.000 | 5.000 | 10.000 | 10.000 | 10.000 | 20.000 | 20.000 | 30.000 |
| Serv. gestión contable | | 500 | 500 | 500 | 500 | 500 | 500 | 500 | 500 | 500 | 500 | 500 | 500 |
| **SUBTOTAL** | 80.000 | 114.000 | 114.000 | 114.000 | 120.500 | 120.500 | 122.500 | 127.500 | 127.500 | 132.500 | 142.500 | 147.500 | 162.500 |
| Pago dividendos | 0 | 0 | 0 | 0 | 375 | 938 | 1.875 | 3.750 | 7.500 | 15.000 | 22.500 | 30.000 | 37.500 |
| **TOTAL PAGOS** | 80.000 | 114.000 | 114.000 | 114.000 | 120.875 | 121.438 | 124.375 | 131.250 | 135.000 | 147.500 | 165.000 | 177.500 | 200.000 |
| **Situación efectivo fin mes** | 920.000 | 806.000 | 692.000 | 578.000 | 459.625 | 344.438 | 232.563 | 126.313 | 41.313 | -6.188 | -21.188 | 1.313 | 51.313 |

A continuación se detalla el origen de algunos de los valores expuestos:

- **Sueldos:** Para el pago de honorarios se tuvieron en cuenta los siguientes perfiles de empleados:
  - Desarrollador Senior
  - Desarrollador Junior
  - QA Junior
  - Diseñador Gráfico/Web Semi-senior
  - Vendedor/comercial

- **Alquiler + servicios:** se alquilarán puestos de trabajos en una oficina de co-working. El pago por puesto de trabajo incluye el costo de los servicios, como internet, luz, etc.

- **Infraestructura GoogleCloud:** se utilizará una infraestructura de servidores en la nube por las ventajas y facilidades que da para crecer a la medida de los requerimientos del negocio. Se puede ver que el monto expuesto va evolucionando. En un principio se requiere un servidor con características básicas para el desarrollo. Luego, conforme se vayan adquiriendo nuevos usuarios, se irán sumando servidores, procesadores, memoria, etc.

- **Publicidad:** se invertirá en campañas de publicidad en internet. El monto destinado a publicidad irá creciendo paulatinamente conforme se vayan adquiriendo nuevos ingresos.

- **Inversión inicial:** para poder comenzar con la actividad es necesario contar con un capital para emprender. Se requiere un monto inicial de $1.000.000 que posibilite financiar los costos de los primeros 12 meses de actividad de la empresa. Un 10% del monto se aportará con capital propio y se buscará inversión de terceros (particulares, incubadoras o aceleradoras) por los $900.0000 restantes.

- **Pago de dividendos:** se contabiliza una participación en los dividendos por parte de los inversores de un 15% de los ingresos.

- **Ingresos por venta de licencias:** para contabilizar los ingresos se tuvo en cuenta el patrón de ingreso por medio de venta de licencias. La idea es que luego del primer año de actividad se alcance el objetivo de venta de 1.000 licencias pagas con un valor aproximado de US$ 15 mensuales cada una (en un plan básico). Esto permitirá que luego de un año se comience a facturar un monto aproximado de US$15.000 mensuales que posibilitarán el autofinanciamiento a futuro de las actividades del emprendimiento.

# 23. Administración del proyecto

## 23.1. Gestión de configuración

La Gestión de la Configuración del Software (GCS) es un método que permite controlar el desarrollo de software y sus modificaciones a lo largo de todo su ciclo de vida. Busca identificar claramente los ítems desarrollados y sus distintas versiones para que, cuando ocurra un cambio, contemos con la trazabilidad necesaria para conocer sus orígenes y si es necesario volver a versiones anteriores (Koskela, 2013);

La planificación e implementación de la GCS constituyen un factor clave para alcanzar la calidad del software y posibilitar su mantenimiento a lo largo del tiempo. Esta tarea no debería ser subestimada ni ignorada, ya que de ello puede depender el éxito o fracaso del producto desarrollado (Koskela, 2013).

Koskela (2013) menciona que en la metodología Scrum no se hace referencia a la GCS ni se dan pautas o reglas específicas a seguir. No obstante, el mismo autor recuerda que Scrum es una metodología flexible que posibilita el uso de otras herramientas, métodos y técnicas que se crea necesario en pos de alcanzar los objetivos. Por lo tanto resulta conveniente implementar algunos conceptos de GCS como el versionado del código y la gestión de cambios, pero siempre tratando de que la documentación no torne demasiado pesado el trabajo ni entorpezca el proceso

general de desarrollo. El autor menciona que la GCS puede ser menos abundante y no tan formal en proyectos pequeños o con equipos de trabajo con pocos integrantes donde la comunicación es más directa y fluida.

A continuación se describirán algunas herramientas y estrategias que serán utilizadas para la gestión de la configuración en este proyecto.

Se aclara que algunas de estas ideas o prácticas están pensadas para cuando el producto crezca y se incorporen nuevas personas al equipo de desarrollo.

### 23.1.1. Control de versiones

Las herramientas utilizadas para el versionado de código fuente se suelen denominar CVS, por sus siglas en inglés de Concurrent Versions System.

Este tipo de sistemas permite el acceso concurrente a los distintos archivos que constituyen el código fuente del programa por parte de los integrantes de un equipo de trabajo. Trabaja con una arquitectura de tipo cliente-servidor en el que el servidor se encarga de mantener organizadas las distintas versiones del código del producto y cada cliente posee una copia de alguna de las versiones sobre las que desee trabajar.

Para este proyecto se utilizará GIT, una herramienta de versionado de código gratuita. Como servidor para almacenar el código versionado se utilizará una cuenta gratuita en la plataforma GitLab. Esta plataforma es ampliamente difundida ya que permite la creación de repositorios de código privados.

Se crearán 3 repositorios GIT para almacenar el código fuente de PranaCMS:

- Repositorio para la API REST;
- Repositorio para la aplicación Backend;
- Repositorio para la librería cliente.

Además, se creará un cuarto repositorio para almacenar todo lo referido con el prototipo y diseño de las interfaces.

### 23.1.2. Nomenclatura de versiones

Se utilizará el estándar SemVer (Semantic Versioning) para nomenclar las distintas versiones del software producido.

Este estándar utiliza una nomenclatura del tipo X.Y.Z, donde X, Y, Z son números enteros positivos que se utilizan para enumerar las versiones del software.

Cada vez que producimos y liberamos una nueva versión del sistema debemos modificar uno o más de estos números de acuerdo a uno de los siguientes casos:

- **Patch:** se denomina patch o bugfix a una modificación en el código fuente con el propósito de corregir algún error o falla. Esta corrección debe mantener la compatibilidad con la API actual. Al realizar esta modificación incrementamos en uno el valor de Z.
- **Minor:** representa una modificación en el código que añade una nueva característica o funcionalidad o que modifica algún comportamiento pero siempre manteniendo la compatibilidad con la API actual. Se debe incrementar en uno el valor de Y, y volver Z a cero.
- **Major:** incrementamos el valor de X cuando realizamos una modificación en el código fuente que modifica la API actual, por lo que ya no será compatible con versiones anteriores. Esto obliga a que el usuario que integre la API deba realizar la adecuación pertinente en sus programas. Al incrementar X, debemos reiniciar Z e Y a 0;

El versionado iniciará con la numeración 0.0.0. El valor 0 en el lugar de X indica que la API aún está en etapa de desarrollo, por lo que la misma aún no se considera estable y es susceptible de cambio en cualquier momento.

Para no confundir a los usuarios, el código fuente de los 3 subsistemas antes mencionados (API REST, Backend y librería cliente) tendrán siempre el mismo valor de X (Major). Por ejemplo, por más que sólo la aplicación Backend sufriera modificaciones importantes y se deba incrementar su numeración X de 1 a 2, entonces se incrementará la versión major de los otros 2 subsistemas para mantener la sincronía del producto como un todo. Los valores de Patch y Minor sí podrán evolucionar por separado.

### 23.1.3. Planificación de versiones

Cada historia de usuario a desarrollar tendrá asociada un número de versión. Esto permitirá a futuro que si existen varios desarrolladores trabajando en el producto puedan hacerlo en forma concurrente sin afectar el trabajo del resto del equipo.

Al iniciar el trabajo de una historia de usuario se creará una rama en el código fuente con el número de versión asociada a la misma y una pequeña descripción literal del nombre de la historia. Por ejemplo versión 0.12.0 – Permitir cambio de la clave de una aplicación.

Para el caso de correcciones o bugfix que no dependen de una nueva historia de usuario se crearán tareas simples que también tendrán asociadas un número de versión. Por ejemplo versión 0.12.3 – Validar longitud máxima para el cambio de clave de una aplicación.

## 23.2. Gestión de tareas y cambios

La gestión de tareas y cambios en Scrum se realiza mediante el uso de la Pila de Producto o Product Backlog. En esta pila añadiremos las historias de usuarios con las nuevas funcionalidades o cambios a implementar y las distintas tareas menores que surjan de correcciones o bugfix.

Esta pila mantiene el registro de todo lo que está pendiente de realizar a futuro en relación al producto.

En Scrum, al comienzo de cada Sprint, el Project Manager decide en conjunto con el Product Owner qué historias o tareas de esta pila serán desarrolladas en el próximo Sprint. Con esta lista de tareas se crea la Pila de Sprint o Sprint Backlog que contiene todas las historias o tareas que deberán estar desarrolladas para el cierre del Sprint. Luego, cada desarrollador del equipo toma tareas de esta pila y va cambiando su estado.

Tanto la Pila del Producto como las Pilas de Sprints se gestionan en tableros que se componen de distintas columnas con los estados por lo que pasa la tarea o historia hasta considerarse terminada y aprobada.

Para la construcción de Pilas y tableros se utilizará Trello, un software online y gratuito que permite este tipo de trabajos de una forma simple e intuitiva.

## 23.3. Estimaciones y métricas

En Scrum todas las tareas e historias de usuario deben tener por lo menos asociadas una cantidad de horas estimadas para concretar las mismas.

Una tarea debe ser estimada antes de comenzar su desarrollo. Dicha estimación no siempre suele llevarse a cabo por la misma persona que realiza su codificación. Estos factores influyen en que la cantidad de horas trabajadas difieran de la cantidad de horas estimadas.

Es por esta razón que se confeccionará una métrica general que permita visualizar al cierre de cada Sprint la diferencia entre la cantidad de horas reales sobre las estimadas. Mantener esta métrica ayudará a ir ajustando a lo largo del proyecto las estimaciones realizadas.

A medida que se incorporen nuevos desarrolladores al equipo esta métrica también puede llevarse a nivel personal por cada desarrollador para detectar posibles desfasajes de acuerdo a las capacidades técnicas de cada uno.

También se implementarán diagramas Burndown por cada Sprint que permitan conocer día a día el avance realizado sobre las tareas e historias. Este diagrama relaciona el tiempo total estimado necesario para completar las tareas del Sprint con

el progreso real que se va logrando día a día sobre las mismas. Ésta es una métrica viva que puede ir variando conforme se re-estiman las tareas ya sea incrementando o reduciendo su valor.

Scrum también maneja el concepto de Story Point, que es un puntaje relativo (y subjetivo) que se le da a una historia de usuario para caracterizar su valor para el negocio del cliente. Muchas veces suele guardar relación con el tamaño o complejidad de la tarea.

Realizar una métrica entorno a este valor resulta interesante cuando hay más de un desarrollador en el equipo de trabajo. Por ejemplo, se puede construir una métrica que nos indique al final de cada Sprint cuántos Story Points pudo completar cada desarrollador o todo el equipo de trabajo. Si los puntajes están bien asignados esta métrica puede ser un claro indicador de la productividad de una persona o equipo.

A su vez, también se puede realizar una métrica que indique el puntaje promedio de Story Points resueltos por cada desarrollador o equipo por día, semana o Sprint.

Por último, también se implementarán métricas que relacionen la cantidad de tareas de tipo bugfix (o corrección de errores) que surjan por cada historia desarrollada. Esto permitirá conocer a futuro cuáles son los puntos más críticos del sistema sobre los cuáles se debería profundizar el análisis y enfocar las tareas de testing. Además esta métrica podría computarse también por cada desarrollador, lo que ayudaría a encontrar problemas en el desempeño o al menos dar pautas para indagar las causas que ocasionaron dichos errores en el producto (por ejemplo puede que el problema no sea el desarrollador en sí sino la forma en la que se transmiten los requerimientos o la falta de capacitación sobre una tecnología o funcionalidad).

## 23.4. Administración del riesgo

Según Corral González (2007) en Scrum se sustituye la gestión del riesgo explícita por la gestión del riesgo continua en el sentido que es prioritario detectar y resolver diariamente cualquier impedimento que los miembros del equipo hayan relevado. Estos impedimentos o riesgos son gestionados por el Scrum Master a través de una Pila de Impedimentos o Impediment Backlog. El objetivo es eliminar cualquier obstáculo que frene o impida el avance de otras tareas.

Este relevamiento diario se realiza durante la reunión diaria de proyecto o Daily Meeting en la que lo miembros del equipo deben hacer notar si detectan algún inconveniente que afecte al proyecto en general o al Sprint actual.

El mismo autor, además, proporciona una lista de riesgos comunes en todos los proyectos y cómo Scrum sirve como herramienta para mitigar su ocurrencia. Se agrega una columna adicional en la que se da pautas sobre posibles acciones de contingencia para cada caso:

Tabla 13

*Administración del riesgo en la metodología Sprint*

| Riesgo | Mitigación en Scrum | Acciones de contingencia |
|---|---|---|
| **Ampliación descontrolada de características** | En Scrum se parte de un Product Backlog donde se listan de manera genérica todas las características que abarcará el producto. Estas características son revisadas y priorizada por el Product Owner. Durante el transcurso del proyecto se pueden realizar cambios o agregar funcionalidades. Si esto sucede es aceptable que el alcance del proyecto cambie y pueda crecer el número de Sprints y modificar los tiempos de entrega. La ampliación del alcance es controlada y avalada por el cliente. | Re-priorizar junto al Product Owner el orden de las tareas. Re-estimar con el equipo las nuevas tareas o las que fueron modificadas. Analizar el impacto del cambio y re-planificar Sprints. Comunicar al cliente cualquier modificación en las fechas de entrega para obtener aprobación. Si el cronograma no se puede correr o el presupuesto no puede variar negociar características secundarias para hacer lugar a las características principales. |
| **Captura de requisitos mal realizada** | Scrum reduce el riesgo de realizar una captura demasiado exhaustiva (sólo se listan las historias de usuario) y realiza el detalle correspondientes al desarrollar la historia durante un Sprint. Si existe un problema o falta información esto puede ser reportado tempranamente durante una Daily Meeting. | Enfatizar en la captura de impedimentos en la Daily Meeting por parte del equipo. Si el equipo tiene experiencia intentar anticipar estos problemas durante el Sprint Planning. Durante el Sprint Retrospective poner foco en por qué se escapó este problema para tratar de que anticiparse en futuras situaciones similares. |
| **Calidad insuficiente** | Durante el Sprint Review el Product Owner podrá detectar de manera temprana problemas de usabilidad, tiempos de respuesta inadecuados, entre otros. Pueden existir, sin embargo, problemas que permanezcan ocultos hasta que no se hagan pruebas reales en ambientes productivos. | Implementar revisión de código por parte de miembros más experimentados del equipo. Implementar tests automatizados. Realizar pruebas con datos obtenidos de entornos productivos. |

| | | |
|---|---|---|
| **Plazos optimistas impuestos** | En Scrum el equipo es quien estima y/o re-valida las estimaciones en conjunto. Esto ayuda a plantear escenarios de estimación más acordes a la realidad del equipo de trabajo. | Estar atentos a brechas grandes entre lo estimado y lo realmente trabajado. Analizar el origen de estas desviaciones durante el Sprint Retrospective. Analizar en conjunto con el equipo mecanismos para evitar o reducir estos desfasajes. |
| **Diseño inadecuado** | Estos problemas pueden manifestarse durante las Daily Meeting, durante un Sprint Review o durante el Sprint Retrospective. Mientras antes se detecte el problema menor será el impacto en lo planificado. | Implementar revisión de pares por miembros del equipo más experimentados para detectar y corregir problemas antes de pasar las tareas a resueltas. |
| **Personal inadecuado** | En Scrum se trata de que los miembros de los equipos sean interdisciplinarios. Se fomenta el trabajo en equipo, la capacitación constante y la transferencia de conocimiento de manera diaria. | Asegurarse que en el equipo se cuente por lo menos con algún profesional experto en las áreas más críticas. Realizar revisión de pares. Permitir el intercambio de conocimientos durante el Sprint Retrospective. Fomentar la capacitación constante del equipo en las aptitudes que se consideren críticas para el éxito del proyecto. |
| **Fricción con los clientes** | Scrum pondera la colaboración activa con el cliente. El cliente puede delegar un Product Owner para supervisar, agilizar y encaminar el desarrollo de manera diaria. El cliente es visto como un miembro más del equipo. El equipo trabaja en pos de lograr los objetivos del cliente. | Fomentar una participación más activa con el Product Owner. Insistir en la realización de Sprint Reviews y de obtener la aprobación de lo mostrado. Solicitar feedback en formato de correcciones o propuesta de mejoras. |

# 24. Conclusiones

En el presente Trabajo Final de Graduación pude aplicar gran parte de los contenidos aprendidos a lo largo de la carrera.

Esta instancia académica tuvo por objeto el desarrollo de un proyecto que concluyo con el desarrollo de un producto de software. Todo el proceso fue documentado en el presente texto y contempló las actividades requeridas en el ciclo de vida de un software, comenzando desde las etapas iniciales de relevamiento, análisis, diseño, gestión y finalmente construcción y pruebas.

Esta tarea requirió de mucho esfuerzo y aprendizaje, los cuales permitieron profundizar en los conocimientos necesarios para abordar la problemática a través de la elaboración de un Marco Teórico. Las herramientas ingenieriles adquiridas ayudaron a completar todas las actividades planificadas en tiempo y forma.

Como resultado de este largo proceso se obtuvo un sistema informático que puede ser de gran utilidad y beneficiar a los potenciales usuarios para los cuales fue dirigida la propuesta.

Agradezco la guía de los tutores a lo largo del proyecto, los cuales no sólo me encaminaron sino que también me motivaron para poder alcanzar la meta.

Me encuentro muy satisfecho a nivel personal con el resultado obtenido y contento de haber podido transitar este largo camino para convertirme en Ingeniero en Software.

## 25. Bibliografía


*Acerca de Node.js*. (s.f). Recuperado el 28 de 03 de 2017, de https://nodejs.org/es/about/

Alfonseca Moreno, M., de la Cruz Echeandía, M., Ortega de la Puente, A., & Pulido Cañabate, E. (2006). *Compiladores e intérpretes: teoría y práctica.* Madrid: Pearson Educación.

Andreu, J. (2011). *Gestión de servidores web (Servicios en red).* Madrid, España: Editex, S. A.

Arias, A. (2015). *Desarrollo Web con CMS. Drupal y Joomla* (2da ed.). Galicia: IT Campus Academy.

Aubry, C. (2011). *WordPress 3: un CMS para crear su sitio Web.* Barcelona, España: Ediciones ENI.

Bas Abad, V. J. (18 de Septiembre de 2015). *Estudio comparativo de BBDD relacionales y NoSQL en un entorno industrial.* Recuperado el 04 de Abril de 2017, de https://riunet.upv.es/bitstream/handle/10251/55530/Memoria.pdf?sequence=1&isAllowed=

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., . . . Thomas, D. (2001). *Manifiesto por el Desarrollo Ágil de Software*. Recuperado el 05 de 04 de 2017, de http://agilemanifesto.org/iso/es/manifesto.html

Blé Jurado, C. (Enero de 2010). *Diseño Ágil con TDD.* Obtenido de http://www.carlosble.com/downloads/disenoAgilConTdd_ebook.pdf

Centro de Apoyo Tecnológico a Emprendedores. (13 de 07 de 2012). Recuperado el 21 de 03 de 2017, de https://www.bilib.es/fileadmin/estudio_sistemas_gestion_contenidos_web_cms.pdf

Conesa Caralt, J., Rius Gavidia, Á., Ceballos Villach, J., & Gañán Jiménez, D. (2010). *Introducción a .NET.* Barcelona, España: Editorial UOC.

Corral González, R. (28 de Junio de 2007). *Exprimiendo Scrum: Scrum y la gestión del riesgo*. Recuperado el 12 de Junio de 2017, de https://geeks.ms/rcorral/2007/06/28/exprimiendo-scrum-scrum-y-la-gestin-del-riesgo/

Durán, F., Gutiérrez, F., & Pimentel, E. (2007). *Programación orientada a objetos con Java.* Madrid, España: Paraninfo.

Eguiluz, J. (2009). *Introducción a CSS.* Recuperado el 14 de Septiembre de 2016, de http://librosweb.es/libro/css/

Eguiluz, J. (2009). *Introducción a JavaScript.* Recuperado el 14 de Septiembre de 2016, de http://librosweb.es/javascript/

Eguiluz, J. (2009). *Introducción a XHTML*. Recuperado el 14 de Septiembre de 2016, de http://librosweb.es/xhtml/descargar/pdf/

Fowler, M. (2011). *Patterns of Enterprise Application Architecture.* Indiana, Estados Unidos: Addison-Wesley.

Groussard, T. (2010). *Java Enterprise Edition: Desarrollo de aplicaciones web con JEE 6.* Barcelona, España: Ediciones ENI.

Henderson, C. (2006). *Building Scalable Web Sites.* Sebastopol, Estados Unidos: O'Reilly Media Inc.

IsYourWeb. (2015). *Comparativa Drupal, Joomla y Wordpress.* Recuperado el 22 de Septiembre de 2016, de http://www.isyourweb.com/comparativa-drupal-joomla-y-wordpress

Korth, H. F. (2002). *Fundamentos de bases de datos* (4ta ed.). Madrid, España: Mc Graw Hill.

Koskela, J. (2013). *Software configuration management in agile methods.* (V. Electronics, Ed.) Recuperado el 10 de Junio de 2017, de http://www.vtt.fi/inf/pdf/publications/2003/P514.pdf

Lledó, P. (2012). *Gestion Agil de Proyectos: Lean Project Management.* Estados Unidos.

Maciá Pérez, F. (2008). *Administración de servicios de Internet: De la teoría a la práctica.* Alicante, España: Publicaciones de la Universidad de Alicante.

Marqués, A. (11 de Abril de 2013). *Conceptos sobre APIs REST.* Recuperado el 09 de 04 de 2017, de http://asiermarques.com/2013/conceptos-sobre-apis-rest/

Martelli, A. (2006). *Python in a Nutshell: A Desktop Quick Reference* (2da ed.). Estados Unidos: O'Reilly Media, Inc.

Mozilla Developer Network. (3 de Septiembre de 2016). *Referencia de Elementos HTML: script*. Recuperado el 2 de Abril de 2017, de https://developer.mozilla.org/es/docs/Web/HTML/Elemento/script

Niño, J. (2010). *Aplicaciones web.* Editex.

Palacio, J. (2014). *Gestión de proyectos Scrum Manager.* Scrum Manager.

Pasquali, S. (2013). *Mastering Node.js.* Reino Unido: PacktPub.

Perry, B. (2004). *Java Servlet & JSP Cookbook: Practical Solutions to Real World Problems.* Estados Unidos: O'Reilly Media, Inc.

Ruano Vázquez, F. J. (2014). *Análisis y Desarrollo de MongoDB y Redis en Java.*

Séculi, M. (2016). *WordPress vs Joomla vs Drupal – Infografía 2016.* Recuperado el 27 de Septiembre de 2016, de http://www.marcosseculi.es/wordpress/wordpress-vs-joomla-vs-drupal/

Sommerville, I. (2005). *Ingeniería del Software 7ma edición* . Madrid: PEARSON EDUCACIÓN. S.A.

Tortajada Cordero, J. (2014). *La guía definitiva de los lenguajes de marcas.* Madrid, España: [s.n.].

Trias, F., de Castro, V., Lopez-Sanz, M., & Marcos, E. (2015). Migrating Traditional Web Applications to CMS-based Web Applications. *Electronic Notes in Theoretical Computer Science, 314*, 23-44.

Valdespino Alberti, A. I., León Rodríguez, K., Díaz Sordo, G., Gómez Mazorra, P., Mailán Andricaín, A., Martínez Ortega, R. M., . . . Ordoñez García, I. I. (2014). Análisis de Sistema de Gestión de Contenidos para una Red Colaborativa en la Facultad de Ciencias Médicas Dr. Enrique Cabrera. *Revista Habanera de Ciencias Médicas, 13*(6), 973-983.

W3 Techs. (2016). *Usage of content management systems for websites.* Recuperado el 27 de Septiembre de 2016, de https://w3techs.com/technologies/overview/content_management/all

# 26. Anexos

## 26.1. Anexo A – Ejemplo de entidades y contenido a almacenar

Se presenta el siguiente ejemplo básico para ilustrar el ejemplo en el que un usuario configura una entidad para administrar noticias.

Luego de configurar la entidad, se almacenaría la siguiente información en la base de datos relacional:

- **Entity:** noticia
- **Attributes:**
  - **Titulo:**
    - **name**: titulo
    - **attribute_type**: text
    - **required**: true
    - **label**: Título de la noticia
    - **order:** 1
    - **display_in_list:** true
    - **multiple:** false
  - **Contenido:**
    - **name:** contenido
    - **attribute_type**: richtext
    - **required**: true
    - **label**: Contenido de la noticia
    - **order:** 2
    - **display_in_list:** false
    - **multiple:** false
  - **Imagen:**
    - **name:** imagen
    - **attribute_type**: image
    - **required**: true
    - **label**: Imagen principal
    - **order:** 3
    - **display_in_list:** true
    - **multiple:** false
  - **Tags:**
    - **name:** tags
    - **attribute_type**: text
    - **required**: true
    - **label**: Etiquetas
    - **order:** 4

- **display_in_list:** false
- **multiple:** true
- **min:** 1
- **max:** 5

Por otro lado, cuando el usuario cargue contenido para esta entidad la misma se almacenará como un documento en una colección de nombre noticia en la base de datos no relacional. El contenido tendría el siguiente formato en notación JSON:

{

_id: “1”,

titulo: “Éste es un título real de la noticia!”,

contenido: “Aquí el usuario habría cargado el contenido de la noticia como texto enriquecido”,

imagen: “path-a-la-imagen.png”,

tags: [“Etiqueta 1”, “Etiqueta 2”, “Etiqueta 3”]

}

**ANEXO E** – FORMULARIO DESCRIPTIVO DEL TRABAJO FINAL DE GRADUACIÓN

**AUTORIZACIÓN PARA PUBLICAR Y DIFUNDIR TESIS DE POSGRADO O GRADO A LA UNIVERIDAD SIGLO 21**

Por la presente, autorizo a la Universidad Siglo21 a difundir en su página web o bien a través de su campus virtual mi trabajo de Tesis según los datos que detallo a continuación, a los fines que la misma pueda ser leída por los visitantes de dicha página web y/o el cuerpo docente y/o alumnos de la Institución:

| | |
|---|---|
| **Autor-tesista**<br>*(apellido/s y nombre/s completos)* | Gallardo, Adrián Marcelo |
| **DNI**<br>*(del autor-tesista)* | 33303396 |
| **Título y subtítulo**<br>*(completos de la Tesis)* | Sistema de Administración de Contenido en la nube con integración en clientes web |
| **Correo electrónico**<br>*(del autor-tesista)* | agallardo@outlook.com.ar |
| **Unidad Académica**<br>*(donde se presentó la obra)* | Universidad Siglo 21 |

Otorgo expreso consentimiento para que la copia electrónica de mi Tesis sea publicada en la página web y/o el campus virtual de la Universidad Siglo 21 según el siguiente detalle:

| | |
|---|---|
| **Texto completo de la Tesis**<br>*(Marcar SI/NO)* | SI |
| **Publicación parcial**<br>*(Informar que capítulos se publicarán)* | |

Otorgo expreso consentimiento para que la versión electrónica de este libro sea publicada en la página web y/o el campus virtual de la Universidad Siglo 21.

**Lugar y fecha:**

_____

_____ _____

**Firma autor-tesista** **Aclaración autor-tesista**

Esta Secretaría/Departamento de Grado/Posgrado de la Unidad Académica: _____certi
fica que la tesis adjunta es la aprobada y registrada en esta dependencia.

_____ _____

Firma Autoridad Aclaración Autoridad

Sello de la Secretaría/Departamento de Posgrado