Universidad Nacional de La Plata - Facultad de Informática

Tesina de Licenciatura en Sistemas

ONE: Sistema para Marketing Digital

Tesista: Agustín Vieta

Director Académico: Laura Fava

Director Profesional: Ariel Héctor Martínez

Junio 2019

ÍNDICE DE CONTENIDO

CAPÍTULO 1. Introducción

1.1 Motivación

El marketing digital es la aplicación de las estrategias de comercialización llevadas a cabo en los medios digitales. Todas las técnicas del mundo offline son imitadas y traducidas a un nuevo mundo, el mundo online. En el ámbito digital aparecen nuevas herramientas como la inmediatez, las nuevas redes que surgen día a día y la posibilidad de mediciones reales de cada una de las estrategias empleadas.

En sus inicios, el marketing online se basaba en las páginas web 1.0 y venía a ser una translación de la publicidad de los medios tradicionales (televisión, radios, medios en papel) a las primerizas páginas web, las cuales no permitían una bidireccionalidad en la comunicación con los usuarios. Las empresas anunciantes controlaban totalmente el mensaje y se limitaban a exponerlo a la audiencia.

Además, la publicidad de la etapa web 1.0 se limitaba, en la mayoría de las ocasiones, a reproducir un escaparate de productos o servicios en forma de catálogo online. Aún así, este tipo de publicidad ya apuntaba interesantes virtudes, como el alcance potencialmente universal, la posibilidad de actualización de los contenidos y la combinación de textos, imágenes y, poco a poco, también del formato multimedia.

Pero en pocos años llegó la revolución. Un frenético desarrollo tecnológico permitió la introducción masiva de un Internet de nivel superior. Nacía la web 2.0 y con ella, el marketing 2.0.

Con la web 2.0 nace la posibilidad de compartir información fácilmente gracias a las redes sociales y a las nuevas tecnologías de información que permiten el intercambio casi instantáneo de piezas que antes eran imposibles, como videos, imágenes, etc. Se comienza a usar Internet no solo como medio para buscar información sino como comunidad, donde hay relaciones constantemente y feedback con los usuarios de diferentes partes del mundo.

En este ámbito se desarrolla el marketing digital, ya que los usuarios pueden hablar de la marca libremente. Ellos tienen un poder importantísimo que antes solo se le permitía a los medios: la opinión.

Las técnicas de comercialización debieron cambiar su paradigma. Si antes los distribuidores, los medios, y los productores eran los que tenían el poder de la opinión, ahora el foco está en el usuario. Este es capaz de buscar aquello que quiere gracias al poder de los search engines (Google, Yahoo, Bing, etc.), y no solo preguntar a los medios dados si su decisión es correcta, también tiene la posibilidad de leer reseñas, comentarios y puntuaciones de otros usuarios. Es por eso que una estrategia digital debe incluir todos los espacios relevantes en donde el target interactúe, buscando influenciar opiniones y opinadores, mejorar los resultados de los motores de búsqueda, y analizando la información que estos medios provean para optimizar el rendimiento de las acciones tomadas. El marketing digital es el conjunto de diseño, creatividad, rentabilidad y análisis buscando siempre un ROI[1]. (Burgos *et al.*, 2009).

Este contexto motiva el desarrollo de una plataforma inteligente que permita centralizar el manejo de campañas publicitarias sobre las distintas plataformas digitales existentes. Esto permitirá a los usuarios lanzar una campaña multiplataforma desde un único sitio.

1.2 Objetivos

El objetivo de esta Tesina es desarrollar una aplicación web para una empresa de Marketing Digital que facilite la creación de campañas publicitarias para distintas plataformas como Twitter, LinkedIn o Spotify. Esta aplicación deberá facilitar la difusión (de los mensajes) de las campañas de marcas líderes de América Latina a las audiencias correctas, minimizando el trabajo interno que involucra la creación y lanzamiento de una nueva campaña en las redes sociales. El éxito de esta aplicación, denominada ONE, radica en ofrecer una experiencia de usuario superior a la de las plataformas líderes del mercado para lograr la aceptación de

[1] Return On Investment, valor económico generado como resultado de la realización de diferentes actividades de marketing

los clientes, ya sean agencias publicitarias o anunciantes directamente.
Lo novedoso de ONE es que estará desarrollada íntegramente en JavaScript, una API RESTful en Node.js y un frontend en React.

1.3 Estructura de la Tesina

En el segundo capítulo de esta tesina se abordarán los aspectos principales del marketing digital, se analizarán las distintas herramientas que brinda y se expondrán las principales redes sociales junto a sus sistemas de publicidad.
En el tercer capítulo se expondrá la problemática de la empresa para la cual se desarrolló ONE. Se detallarán los actores y tareas involucradas en el actual proceso de carga de campañas y se abordarán las características principales del sistema.
En el cuarto capítulo se analizarán las tecnologías principales que estuvieron involucradas en el desarrollo de ONE. Se describirán y analizarán las características más importantes de cada una de ellas.
En el quinto capítulo se mostrará la arquitectura de ONE. Además se expondrá el trabajo realizado en materia de diseño, como la interfaz de usuario creada, y la experiencia de usuario, pilares muy fuertes en el desarrollo de ONE. Se presentarán los primeros bocetos del sistema y como fue la evolución a las pantallas actuales.
En el sexto y último capítulo de esta tesina, se expondrán las conclusiones obtenidas a partir del trabajo realizado y se presentarán posibles líneas de trabajo futuro.

CAPÍTULO 2. Estado del arte

2.1 ¿Qué es el Marketing?

El marketing es una ciencia humana que tiene el objetivo de estudiar el comportamiento del consumidor con relación a los productos y/o servicios. Es decir, la relación entre la oferta y la demanda (Armstrong & Kotler, 2008).

El marketing se basa en 6 axiomas[2] principales:

- **Axioma de las necesidades:** El marketing no crea las necesidades, el consumidor ya las tiene. Lo que hace el marketing es usar las necesidades de los consumidores para crear deseos y así satisfacer una necesidad.
- **Axioma del marketing mix:** Para lograr un buen desempeño del marketing es preciso que los elementos del marketing mix (Precio, Plaza, Producto y Promoción) estén en correcto equilibrio. Si una venta no se realiza es porque uno de los elementos del marketing mix no se adecuan a la demanda y al mercado. Entonces deben cambiarse los elementos del marketing mix y adecuarlos al mercado según convenga a la empresa.
- **Axioma del Valor:** A la hora de adquirir un producto el consumidor no compra el producto en sí. Compra una solución a sus necesidades el cual es de suma importancia para su bienestar emocional.

  Cuando va a comprar un despertador no compra el despertador, compra algo que le avise a llegar a tiempo a su trabajo. Cuando compra un jugo no compra el jugo, compra una solución para su sed.
- **Axioma de la imagen:** El producto se convierte en una imagen en la mente del consumidor por medio del posicionamiento y de esta manera compiten las imágenes, lógicamente hablando lidian las percepciones y no las realidades.
- **Axioma del posicionamiento:** Un producto que no es posicionado por la empresa en el mercado, el mercado se encarga de posicionarlo. Más claramente hablando, si la empresa no logra el posicionamiento del

[2] Premisa que se considera evidente y se acepta sin requerir demostración previa.

producto mediante el marketing mix, y el producto es bueno, el mercado se encargan de posicionarlo hablando del mismo de boca en boca.

- **Axioma del costo:** Comprender el costo que representa para el consumidor satisfacer ese deseo o esa necesidad. El consumidor ahora decide donde comprar sin importar el precio ni que el producto que busca sea más barato en el otro extremo de la ciudad que en la tienda más cercana a su casa; por lo tanto, se requiere estudiar los costos asociados (tiempo y esfuerzo) desde el momento en que se produce hasta que se adquiere el producto. Ahora lo que importa es comprar al menor costo no al menor precio.

2.2 Marketing y la web 2.0

Hace más de 10 años, a través de un diálogo en Internet, nació lo que algunos conocen como el libro por excelencia que define el ocaso de la empresa tradicional, de la organización tal y como se la a conocido hasta la fecha: el **Manifiesto Cluetrain** (Levine, 2000). Cinco personas habían iniciado una conversación en Internet tratando de explicar y entender cómo debería construirse y formarse una nueva forma de enfocar la relación de las empresas con sus clientes. A partir de esa conversación nace el Manifiesto y las 95 tesis que forman su base; todas ellas partiendo de una misma premisa: “los mercados se forman de conversaciones”.

Sin saberlo, seguramente sin buscarlo, detrás de muchas de estas tesis se esconden las bases y los principios de lo que se conoce como “del marketing 1.0 al marketing 2.0”. Un enfoque centrado en personas; un enfoque centrado en el mensaje por encima de la imagen; un enfoque centrado en la experiencia por encima del producto; un enfoque centrado en cómo las empresas, conversando con sus clientes, son capaces de ofrecer una respuesta adecuada a las necesidades de estos.

Tuvo que venir, años más tarde, Tim O’Reilly para ayudar a dar un empujón a todas estas tesis cuando, buscando un nombre para una conferencia relacionada con nuevos modelos y formas de trabajar en Internet, creó el concepto de Web

2.0. O'Reilly creó este concepto para referirse a una segunda generación en la historia de la Web basada en comunidades de usuarios y una gama especial de servicios, como las redes sociales, los blogs o los wikis, que fomentan la colaboración y el intercambio ágil de información entre los usuarios.

2.3 Introducción al marketing digital

Según Calvo Fernandez y Reinares Lara (2001), el marketing digital pone a nuestra disposición una serie de herramientas de gran diversidad con las que pueden realizarse desde pequeñas acciones a prácticamente costo cero hasta complejas estrategias, obviamente más costosas, en las que se pueden combinar infinidad de técnicas y recursos. En este contexto surgen las distintas técnicas y recursos de marketing digital, las cuales se describirán a continuación.

2.3.1 Publicidad display

Es la herramienta de marketing digital más conocida y tradicional. Puede considerarse la valla publicitaria del medio digital. Se trata de anuncios (banners) de diferentes tamaños y formatos (textos, imágenes, gráficos, vídeos) que ocupan un espacio en los sitios de Internet de una forma atractiva y llamativa.

Durante los primeros años de vida, el desarrollo de la publicidad en Internet fue muy básico, abarcando un limitado abanico de formatos en el que predominaban los anuncios de texto. No fue hasta octubre de 1994 cuando la compañía de comunicaciones AT&T lanzó el primer anuncio gráfico en la red, lo que hoy se conoce como banner.

La Fig. 1 muestra un anuncio algo rudimentario que se mostraba en el sitio web hotwired.com y el mensaje decía: "¿Alguna vez su mouse ha hecho clic aquí? ¡Pues debería hacerlo!".

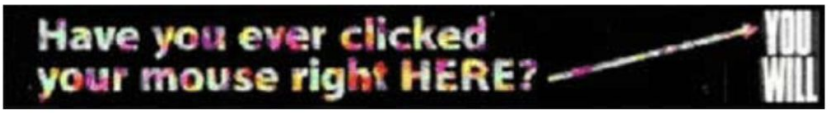

Fig. 1: aviso del sitio hotwired.com.

En sus primeros pasos, el banner cumplía con el concepto tradicional de una publicidad 1.0 impersonal, estática y unidireccional, al igual que un anuncio en prensa o revista. En poco tiempo el banner pasó a convertirse en el formato predominante, inundando como un tsunami gran cantidad de portales y sitios web sin ningún tipo de control. Ante esta situación, surgió en 1996 el Interactive Advertising Bureau (IAB), una institución cuya misión es regular todas las actividades que se pueden desempeñar en la red mediante estandarizaciones y recomendaciones oficiales.

Durante los siguientes años se experimentaron una serie de mejoras a nivel técnico y de infraestructuras, que se tradujeron en un mayor rendimiento de los ordenadores, mayor ancho de banda, nuevas tecnologías de programación, lo cual permitió crear anuncios cada vez más sofisticados, incluyendo audio y video.

A medida que evolucionaba la tecnología y la publicidad, los usuarios también se fueron adaptando paralelamente a las nuevas posibilidades que ofrecía Internet. De esta manera se comienza a hablar de una nueva etapa protagonizada por la publicidad 2.0, una publicidad más social, cooperativa e interesante.

En esta nueva etapa de madurez el banner no goza de gran reputación debido principalmente al desgaste del formato y al auge de nuevas formas publicitarias, que como le ocurrió en su día al banner, le han ganado bastante terreno por la novedad, la efectividad, la versatilidad. No obstante, el banner sigue ocupando la portada de muchos sitios webs como principal fuente de ingresos.

2.3.2 Email marketing

Esta herramienta es una de las más antiguas pero todavía eficaz por haber sabido adaptarse a los cambios y su capacidad de trabajar en combinación con otras estrategias, como el seguimiento y la maduración de clientes.

Utilizado correctamente, el email marketing es una técnica extremadamente potente y efectiva para construir relaciones con los clientes basadas en el valor y la confianza.

Es importante saber que existen dos modalidades diferentes de email marketing. La primera es aquella en la que se envían promociones de forma masiva a

personas que en ningún momento solicitaron recibirla y la segunda son aquellas campañas que se envían a personas que dieron su permiso para ser contactadas por email.

La primera de estas modalidades es lo que se conoce como spam. El envío de estas campañas garantiza la destrucción de la reputación y legitimidad de cualquier empresa. La regla nº1 y más importante en email marketing es no enviar campañas a aquellos que no te han dado su permiso para hacerlo.

La segunda modalidad, el email marketing de permiso, es utilizada diariamente y de forma efectiva por miles de empresas para aumentar el valor de su marca, incrementar las ventas y construir una mejor relación con sus clientes. La diferencia es que estas empresas están enviando sus campañas a personas que sí han solicitado recibirlas. Existen diversas maneras de obtener el permiso de tus suscriptores:

- **Doble Opt-in**. Con este sistema el usuario solicitará su registro en el listado y se le envía un email de confirmación de su identidad. Hasta que ese email no sea confirmado, no pasará a formar parte de la lista. Es la mejor opción para construir una lista de calidad.
- **Opt-in.** En esta ocasión el usuario solicita su registro en el listado pero no ha de confirmar su registro.
- **Opt-out.** En este caso el usuario no solicita su registro. Se le añade a la lista y luego se le envía un email con instrucciones de cómo darse de baja. Muy cerca de ser spam.

Cada vez es más complicado atraer la atención de los consumidores y la mejor manera de dar a conocer productos o servicios es aportando valor al mensaje.

En el caso del email marketing los axiomas de valor e imagen se cumplen a la perfección. Para construir una sólida base de suscriptores que den su permiso para recibir campañas de email o newsletter, es fundamental que se aporte una buena dosis de valor, ya sea mediante la producción de contenidos que sean de interés para el lector o con descuentos especiales en una gama de productos que pueden interesarles al suscriptor.

Es precisamente esta combinación de valor más permiso lo que hace que el email marketing tenga un altísimo porcentaje en su ratio de conversión por visitante y permita desarrollar y fortalecer la relación con los clientes, pudiendo transformar a un comprador casual, en un cliente fiel y evangelizador de la marca.

2.3.3 Buscadores

Los buscadores son herramientas que permiten a los usuarios de Internet encontrar contenidos relacionados con lo que están buscando. Se han convertido en evaluadores de la reputación online de los sitios web. Cuando se introduce un término de búsqueda, el buscador decide, acudiendo a su índice, en qué orden se mostrarán los distintos resultados que contienen los términos introducidos, discriminando entre las páginas almacenadas cuáles son los mejores resultados. Este proceso de selección se realiza mediante algoritmos automatizados, por lo que el buscador puede ofrecer respuesta a una cantidad prácticamente ilimitada de cuestiones, siempre y cuando alguien haya publicado en Internet algo al respecto.

Los buscadores soportan el coste de mantener sus infraestructuras gracias a los resultados publicitarios (o resultados pagados) que muestran además de los resultados de búsqueda convencionales (denominados resultados orgánicos).

Dado que en Internet la búsqueda es una tendencia fundamental, el posicionamiento de una empresa en los buscadores es un activo muy valioso. (Solís, 2016). El SEM o Marketing de Buscadores es la parte del marketing que se ocupa de garantizar la presencia de una empresa en los buscadores. Su objetivo fundamental es que aquellos potenciales clientes que hagan búsquedas relacionadas con el ámbito de actividad de la empresa encuentren resultados que les lleven al sitio web de la empresa.

El SEM engloba todos los esfuerzos dirigidos a tener presencia en los buscadores, tanto en el espacio dedicado a los resultados pagados o publicitarios como en el espacio de los resultados orgánicos o naturales. Por otra parte, se denomina SEO a las técnicas que se utilizan para mejorar la posición de la empresa en los resultados orgánicos de los buscadores. El SEO es una parte del SEM.

Por simplificar, a menudo se utiliza el término SEM para referirse exclusivamente al posicionamiento de la empresa en los resultados pagados de los buscadores y el término SEO para referirse al posicionamiento de la empresa en los resultados orgánicos.

2.3.4 Web / Blogs

Sin lugar a dudas, una web o un blog son dos de las principales herramientas desde las cuales se puede centralizar una campaña de marketing digital o de inbound marketing[3].

Estos sitios se actualizan periódicamente, recopilando cronológicamente textos o artículos de uno o varios autores, apareciendo primero el más reciente, donde el autor conserva siempre la libertad de dejar publicado lo que crea pertinente.

Existen varias formas de clasificar a las webs o blogs:

- **Personales.** Se trata de aquellos que incluyen la opinión individual de una persona.
- **Temáticos/profesionales.** Especializados en una temática o disciplina (marketing, turismo, política, periodismo), habitualmente escritos por profesionales, a título personal, que escriben sobre temas que conocen y dominan. En este grupo estarán los llamados “líderes de opinión”, o blogs con una amplia difusión y credibilidad.
- **Corporativos.** Se trata de aquellos que pertenecen a una empresa. Estos se puede subdividir en:
    - Corporativos externos, para establecer conversaciones con clientes, socios, proveedores, la competencia.
    - Corporativos internos, cuya función es establecer relaciones internas para que todas las unidades de negocio se impliquen en la estrategia de comunicación empresarial

Existe una evolución de los blogs tradicionales, los cuales se denominan microblogs. El microblogging es una forma de comunicación o un sistema de publicación en Internet que consiste en el envío de mensajes cortos de texto a

[3] Estrategia para atraer usuarios y convertirlos en clientes mediante la generación de contenido de valor y no intrusivo.

través de herramientas creadas específicamente para esta función. Su finalidad es la de explicar qué se está haciendo en un momento determinado, compartir información con otros usuarios u ofrecer enlaces hacia otras páginas web.

Se trata de servicios que conjugan el concepto del blog (diario personal en el que el autor va publicando contenidos en orden cronológico) con el de la mensajería instantánea (sistemas de comunicación que permiten mantener conversaciones en tiempo real en Internet con otros usuarios).

2.3.4 Redes sociales

Estas herramientas digitales no han dejado de crecer y ganar popularidad desde la aparición del marketing digital. Además, han sabido adaptarse perfectamente a los cambios y demandas de los consumidores.

Las redes sociales son espacios muy atractivos para las marcas, sobre todo por el gran volumen de usuarios que reúnen, pero al mismo tiempo puede hacer que la comunicación tradicional no funcione en este entorno.

Casi todas las redes sociales ofrecen espacios publicitarios que son excelentes para algunos modelos, como la afiliación.

Otra opción que muchas redes ofrecen es utilizar las mismas herramientas que tienen a su disposición los usuarios y crear espacios de comunidad dentro de la red en cuestión.

Grupos, foros, páginas de fans, eventos, encuestas, son todas ellas fórmulas que el usuario puede utilizar gratuitamente para conectar con otras personas con quien comparte intereses y que una marca también puede utilizar para conectar con sus clientes.

Ventajas de crear una comunidad de marca en una red social:

- No es necesario el registro previo, el usuario ya lo hizo.
- La comunidad se desarrolla en un entorno en el que ya hay millones de usuarios.
- No es necesario invertir en el desarrollo técnico de las funcionalidades, las aporta la red social.

Desventajas de crear una comunidad de marca en una red social

- No ser propietario de los datos de los usuarios ni poder construir una base de datos propia.
- No poder elegir las funcionalidades que están disponibles pues es decisión de la red social.
- No tener control de la forma en que la marca aparece ante el usuario pues las opciones disponibles son algo limitadas.

Gracias a la evolución de las redes sociales, empieza a surgir lo que se conoce como *appvertising*. Cada vez más redes ofrecen a las marcas la posibilidad de desarrollar aplicaciones que utilicen como plataforma la red y la libertad de construir la funcionalidad que mejor les parezca.

Juegos, concursos, guías y en general aplicaciones que mejoran la experiencia del usuario de la red, porque le ofrecen cosas que por defecto la red no ofrecía, tienen cada día mayor éxito.

2.4 Modelos de contratación en campañas de publicidad online

Alet I Vilaginés (2001) explica los cuatro modelos de contratación que ofrece internet para campañas publicitarias, los cuales pueden aplicarse tanto a anuncios gráficos como a anuncios de texto. La Fig. 2 muestra la relación entre estos cuatro modelos y la llegada a los usuarios.

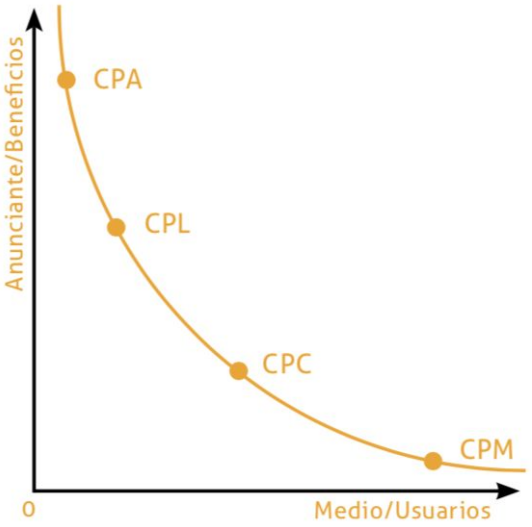

Fig. 2: Relación entre llegada a usuarios y beneficios por usuarios para cada modelo de contratación.

2.4.1 CPM (Cost Per Mille Impressions) - Coste Por Mil Impresiones

Es el modelo más elemental mediante el cual se paga en función del número de impresiones del anuncio, es decir el número de veces que se visualiza la publicidad en una página, independientemente de que los usuarios hagan clic o realicen algún tipo de acción o compra.

Se recomienda este sistema para campañas de branding, es decir, cuando el objetivo sea conseguir visibilidad o reconocimiento de marca, no siendo efectivo para aumentar el beneficio a través de la acción o compra por parte del usuario.

En este tipo de campañas el número de usuarios suele ser muy elevado y, por tanto, el valor aportado por cada uno es menor.

2.4.2 CPC (Cost Per Click) - Coste Por Click

En este modelo se requiere una mínima acción por parte del usuario, únicamente se paga por cada clic que se hace en el anuncio (es el modelo utilizado por Google Adsense en sitios web y Adwords en buscadores), independientemente del número de veces que aparezca o que el usuario termine realizando alguna acción o compra.

Se recomienda este sistema cuando el objetivo sea atraer tráfico hacia una web, con el fin de aumentar el valor generado, ya sea mediante la acción o compra por parte del usuario o la obtención de ingresos por publicidad.

En este nivel, el número de usuarios suele ser inferior al modelo CPM, sin embargo, el beneficio aportado por cada uno es mayor.

2.4.3 CPL (Cost Per Lead) - Coste Por Dirigir o Captar clientes

Conforme se eleva la curva en la Fig. 2, la acción requerida por parte del usuario es mayor. En este modelo, se paga únicamente cuando un usuario hace clic en el anuncio y además realiza algún tipo de acción como el registro de datos mediante formularios, la suscripción a tu boletín electrónico (newsletter) o cualquier otra acción que sea de interés.

Se recomienda este sistema cuando se pretende recabar información acerca de los usuarios, con el fin de convertirlos en futuros clientes. La solicitud de un email o número de móvil suele ser fundamental ya que posteriormente se podrán utilizar estos datos para enviar información u ofertas que se ajusten a las necesidades de cada usuario, habiendo contado previamente con su permiso.

En este nivel, el número de usuarios desciende considerablemente respecto al modelo anterior, sin embargo, el beneficio aportado por cada uno de ellos es muy superior.

2.4.4 CPA (Cost Per Acquisition) - Coste por Adquisición o Compra

Alcanzada la zona más elevada de la Fig. 2, la acción requerida por parte del usuario es todavía aún mayor. En este modelo se paga cuando el usuario realiza una compra en el sitio web, lo que normalmente implica que el usuario ha hecho el recorrido completo: visualizar y hacer clic en el anuncio, rellenar un formulario con sus datos y por último realizar la compra del producto.

Este modelo es muy utilizado por tiendas online ya que el objetivo principal es aumentar la venta de productos mediante la compra por impulso.

El precio de las campañas CPA suele ser el más alto de todos debido a la complejidad de todo el proceso. Además, dependiendo de las condiciones, el sistema de pago al medio suele variar desde un pago fijo por cada venta hasta un porcentaje del valor del producto vendido. A este nivel, son muy pocos los usuarios que llegan y por tanto el valor de éstos es el más alto.

2.4.5 Pago fijo mensual

Este es un modelo adicional, conocido también por patrocinio online, muy diferente a los anteriores en cuanto a su funcionamiento ya que no influyen directamente variables como el número de impresiones, clics o ventas para fijar su precio.

En este caso, pagas una cantidad fija para que tu anuncio aparezca durante un determinado periodo de tiempo en una o varias páginas del sitio web seleccionado. El precio se establecerá en función de las estadísticas y el perfil del sitio web: temática del sitio, nivel de especialización, visitas únicas diarias, número de páginas vistas, etc.

Este tipo de campañas son un complemento ideal al resto de modelos ya que permiten planificar con más fiabilidad al poder comparar entre los distintos modelos y analizar cuál es más eficaz.

2.5 Marketing en redes sociales

Dado que ONE está orientado a realizar campañas publicitarias en distintas redes sociales, analizaremos cómo el marketing impacta sobre ellas.

Internet se ha convertido en una vía de comunicación social. Las redes sociales, lo que se denomina en inglés como *social media*, suponen un nuevo fenómeno de comunicación global, en el que, a diferencia del resto de medios de comunicación, el contenido es creado por la propia audiencia que se agrupa bajo un interés común, compartiendo mensajes, ideas y opiniones.

La mayoría de las redes sociales han tenido un crecimiento espectacular, sobre todo, si se compara con la evolución de otros medios. Para llegar a alcanzar una

audiencia de 50 millones de personas, la radio necesitó 38 años, la televisión 13, Internet 4 y el iPod 3 años. La red social Facebook en tan sólo 9 meses llegó a los 100 millones de usuarios. Hoy supera los 400 millones.

Celaya (2011) describe la importancia de las redes sociales de esta forma:

> Las redes sociales están transformando la manera en que las personas acceden a la información sobre todo tipo de productos y servicios. El nuevo modelo de comunicación online obligará a las empresas a actualizar sus estrategias de marketing y comunicación. Los consumidores ya no quieren limitarse a recibir información sobre un determinado producto o servicio, sino que, además, el usuario quiere formar parte del proceso de promoción del mismo a través de las redes sociales. (p. 45)

2.5.1 Aspectos claves del posicionamiento en las redes sociales

Las redes sociales permiten la participación tanto de personas como de empresas. Para las empresas son un medio ideal ya que les permite entrar en contacto directo con sus clientes, conocer su opinión sobre los productos y servicios ofrecidos, y contar con una información muy directa sobre sus gustos y preferencias. También es una buena vía de conocimiento y comunicación con proveedores y posibles socios o colaboradores.

Pero no se trata de estar por estar. La sola participación no garantiza el éxito. La empresa tiene que definir bien cuál debe ser su mejor estrategia en los medios sociales: ¿qué redes sociales utilizar y cómo administrarlas?, ¿cuál es el perfil de personas a quién desea dirigirse?, ¿cómo va a comunicarse con su comunidad? y ¿con qué frecuencia va participar y qué contenidos y mensajes va a compartir?. Finalmente deberá medir la rentabilidad de su participación. La obtención de resultados no suele ser inmediata.

La actividad de la empresa en las redes sociales debe ser la de interactuar y compartir. La empresa debe nutrir su participación en las redes sociales con información relevante e interesante para sus clientes e interactuar conversando con ellos, escuchando sus opiniones, ideas y comentarios, participando y

contestando a todo ello. Esta participación irá creando una comunidad alrededor de la empresa.

Para algunas empresas, el comportamiento y la forma de actuar en las redes sociales es novedoso e incluso sorprendente y, en ocasiones, puede resultar difícil de entender. Internet pertenece a las personas y las empresas deben establecer una estrategia diferente a la publicidad tradicional. Es sumamente importante familiarizarse con las redes sociales y participar en ellas para poder comprender su funcionamiento y sus posibilidades. La empresa debe cambiar su mentalidad y entender cómo debe dirigirse a los usuarios, el lenguaje que debe utilizar, lo que les puede decir o preguntar, y cómo debe reaccionar cuando el usuario, al que se le ha dado voz y voto, se queja, reclama o pide cosas.

Uno de los grandes retos consiste en encontrar el tiempo suficiente para participar en las diferentes redes sociales. Por esa razón, antes de crear cuentas en varias redes, convendrá analizar si realmente se va a dedicar el tiempo necesario para participar en ellas de una forma activa.

Algunas recomendaciones para actuar en las redes sociales son las siguientes:

- **Centrarse en crear relaciones comerciales y personales, no en vender:** la clave principal de estas redes es justamente la socialización, comunicarse y relacionarse con las personas. No están diseñadas para esgrimir argumentos de venta o para utilizarlas como plataformas donde se incorpore publicidad sobre la empresa o sus productos. La idea primordial es, más bien, utilizar el factor humano para establecer unas relaciones comerciales y personales sólidas que, a largo plazo, puedan aportar resultados económicos.
- **Proporcionar contenido de calidad y con valor añadido:** en las redes sociales se puede compartir mucha información acerca del sector donde se engloba la empresa y, especialmente, el *know how* y conocimientos originales que ésta pueda aportar. Ello ocasionará que los sujetos que representan a la empresa sean considerados como expertos del sector y que la gente comience a interesarse por la información y consejos que la

empresa proporciona. Ese material se puede transmitir de forma escrita o en formato de vídeo, sin reclamar nada a cambio.

- **Buscar contactos de calidad:** conviene expandir de forma progresiva y constante la red de contactos. Sin embargo, convendrá también tratar de identificar a aquellas personas que sean verdaderamente importantes para la empresa. Aquí habrá que incluir a colegas y amigos del sector y, también, a aquellas empresas con las que se pueda llegar a colaborar en un futuro. En este sentido, se debe tener en cuenta de forma prioritaria la calidad de los contactos, antes que la cantidad.
- **Ser auténtico y original:** es imprescindible que la empresa se muestre tal cual es, sin pretender aparentar otra cosa. Se debe fomentar la transparencia y la credibilidad hacia los clientes, contactos y colaboradores. Esa es una de las labores más importantes en las redes sociales, lo cual se puede lograr siendo auténticos y originales.
- **Centrarse en la cuestión de "Cómo puedo servirle":** la mayoría de las empresas buscan en Internet lugares donde poner sus anuncios. En cambio, en las redes sociales, aquellas empresas y personas que centren sus esfuerzos principalmente en ayudar a sus contactos y clientes, tendrán más posibilidades de alcanzar el éxito.

Redondo y Rojas (2017) sostienen que hay que tener en cuenta que las redes sociales se completan con otras herramientas de Internet. Cada una de ellas proporciona un impacto distinto a la web de empresa. Así, por ejemplo, el blog permite obtener relevancia y conocimiento, mientras que una página en Facebook facilita el contacto personal, la cercanía y la interacción con potenciales clientes. A su vez se pueden integrar muy bien los mensajes de Twitter, los vídeos de YouTube y los artículos del blog de la empresa con Facebook o LinkedIn.

2.5.2 Redes sociales generales y sectoriales

Dentro de las redes sociales se pueden distinguir dos tipos: las generales, es decir, aquellas que se dirigen de forma indiscriminada a un público objetivo muy numeroso, y las sectoriales que abarcan a personas con perfiles, intereses,

aficiones, etc. más concretas. Para la empresa, el uso eficaz de ambas redes es importante. (Nieto Churruca y Rouhiainen, 2012).

2.5.2.1 Redes generales

Las más extendidas son las siguientes:

- **Facebook:** es la red social más grande del mundo. Ofrece muchas oportunidades para las empresas.
- **LinkedIn:** se trata de una red social dirigida a profesionales. Es la más recomendable para pymes y empresarios ya que la mayoría de sus usuarios son gente de negocios; a través de ella, se puede conocer a personas nuevas del sector y realizar networking comercial.
- **Xing:** es una red social dirigida a profesionales, similar a LinkedIn, que cuenta con más de 8 millones de usuarios.
- **Twitter:** no es una red social en sentido estricto, sino una página web de microblogging. A pesar de ello, su uso es muy similar al de las redes sociales.
- **Spotify:** tampoco encaja en el concepto de red social, ya que es una aplicación de música por streaming, pero permite seguir personas, compartir playlists, todos los usuarios tienen sus perfiles y se puede ver qué es lo que está escuchando cada uno en tiempo real.

2.5.2.2 Redes sectoriales

Aunque existe gran cantidad de redes sociales sectoriales, cada una de las ellas se centra en una temática diferente. La empresa tendría que conocer cuáles son las redes más activas e interesantes de su sector y participar en ellas. La plataforma más grande de redes sociales sectoriales y grupos de aficionados entorno a un determinado tema es Ning.

2.5.3 Twitter

Twitter es una de las herramientas más eficaces de social media, creada por los programadores Ewan Williams, Biz Stone y Jack Dorsey en el año 2006, permite enviar mensajes cortos, de no más de 280 caracteres a personas que quieren recibirlos. Estas personas se habrán dado de alta como *"seguidores"* de la persona o empresa emisora de tweets (mensajes de Twitter). Los mensajes se pueden enviar y recibir no sólo a través de una PC, también a través de un celular. Esta posibilidad agiliza enormemente el servicio ya que permite estar en contacto con nuestro colectivo de forma inmediata, en cualquier momento y desde cualquier lugar.

Otra gran ventaja que proporciona esta herramienta es su poder de marketing viral. Cuando un seguidor envía o contesta algún mensaje, su mensaje lo reciben a su vez sus seguidores. De esta forma la empresa puede acceder a un enorme colectivo, no sólo al suyo, también a los colectivos de cada uno de sus seguidores, siempre que estos participen.

Sin embargo no es suficiente con participar de cualquier forma. Hay que hacerlo adaptándose a la etiqueta de este potente medio de intercomunicación al que también se le denomina de microblogging. Esto supone, como en cualquier red social, que hay que centrarse en la comunicación entre personas y en el establecimiento de relaciones, evitando utilizarlo como un mecanismo para realizar venta directa.

Las empresas que tienen intención de utilizar Twitter para potenciar su marca, dar a conocer sus productos y crear una comunidad de seguidores pueden utilizar esta herramienta dándole diversos usos:

- Ofrecer servicio de atención al cliente.
- Enviar noticias de actualidad sobre la empresa.
- Mandar noticias relacionadas con el sector o sobre temas de interés.
- Informar sobre ofertas, descuentos o servicios ofrecidos por la empresa.
- Interactuar y conversar con los clientes.

A través de los anuncios pagos es posible amplificar el alcance de contenidos y darle visibilidad estratégica a una cuenta de empresa, promocionándola entre los usuarios cuyas características sean las más adecuadas a sus objetivos de negocio.

Twitter permite realizar campañas orientadas a diferentes objetivos: difundir un nuevo recurso descargable, generar tráfico hacia un sitio web, aumentar la base de seguidores, generar más interacciones en los tweets, promocionar videos, etc.

Los tweets, cuentas y tendencias pagas aparecerán en el feed de los usuarios, y en ocasiones también en la barra lateral, con una leyenda que los identificará como anuncios promocionados.

A la hora de configurar una nueva campaña, los anuncios pueden orientarse a distintos tipos de audiencia según el género, edad, ubicación, idioma, intereses, palabras clave, marcas que siguen, sistemas operativos, tipos de dispositivos y más. Como se ve, la segmentación del público es muy precisa, y brinda muchas opciones que permiten alcanzar a las personas más adecuadas para el negocio.

2.5.4 LinkedIn

LinkedIn tiene alrededor de 467 millones de usuarios en todo el mundo, 106 millones de los cuales participan, al menos, una vez al mes. De estos, el 59% se conectan a través de su celular. ¿Qué significa esto? Es la red social puramente profesional más importante que existe, y sus usuarios activos utilizan con asiduidad su teléfono celular y acceden a LinkedIn a través del mismo.

Las empresas B2B[4] destacan que el 80% de sus leads[5] provienen de LinkedIn, y el 92% de los profesionales de marketing B2B la recomiendan por encima de las demás redes sociales. LinkedIn es una máquina de generar leads, y es la herramienta preferida por los expertos.

En LinkedIn se pueden crear dos tipos de perfil: el profesional y el de empresa, y es de la combinación de los dos, de donde es posible obtener los mayores beneficios para el negocio.

[4] Business to Business, se utiliza como terminología para hablar de la transmisión de información entre fabricantes y distribuidores de un producto.
[5] Se refiere al contacto con un cliente potencial, también conocido como un “prospecto”.

LinkedIn contribuye positivamente tanto en la comunicación interna como externa de las organizaciones, especialmente en el caso de las Pequeñas y Medianas Empresas (Pymes), por lo que éstas deben crear un perfil para dar a conocer sus acciones de negocio a sus grupos de interés, al igual que tener uno para su público interno, con la finalidad de que sus colaboradores tengan fiel conocimiento acerca de quién es la empresa y sus metas a lograr como equipo.

Para empresas grandes que manejan diversos productos con características específicas, y que poseen públicos objetivos con diferentes necesidades, LinkedIn permitirá segmentar sus ofertas y exponer con solidez su portafolio de servicios.

Es recomendable crear grupos de debate relevantes para el negocio a través de esta red social. Al incursionar en la conversación del público, las empresas podrán conocer las perspectivas de otros miembros y construir nuevas relaciones. También podrán compartir contenido propio sobre el sector o la industria en general y hacer comentarios en las publicaciones de los demás usuarios. Esto ayudará a cultivar autoridad, relevancia y reputación, para que los perfiles profesionales o páginas empresariales destaquen y sean consideradas de gran valor.

La publicidad en LinkedIn funciona de forma muy similar a la publicidad online en el resto de redes sociales y buscadores. LinkedIn puede funcionar de dos formas diferentes, detalladas anteriormente:

- CPC
- CPM

Algunos de los tipos de publicidad más populares en LinkedIn son:

- **Anuncios de texto:** anuncios compuestos exclusivamente por un título y una descripción. Son textos muy breves para mostrar al público un producto o servicio.
- **Anuncios de Display:** anuncios en formato de imagen y vídeo, mucho más visuales que los anteriores. El vídeo se autoreproduce al cargar la página y tanto uno como otro dan opción de incluir mucha más información junto con una llamada a la acción.

- **Anuncios dinámicos:** anuncios que se generan en función de un perfil de LinkedIn (gustos, intereses, habilidades), creando un mensaje mucho más personalizado. LinkedIn recomienda este tipo de anuncios para conseguir más seguidores para las páginas de empresa.
- **Contenido patrocinado:** Este tipo de publicidad en LinkedIn es mucho más parecido a lo que estamos acostumbrados a ver en otras redes sociales como Facebook. Se trata de actualizaciones en el muro con el indicativo de "promocionado". Un detalle importante, el contenido patrocinado es el único que se ve en celulares.
- **InMails Patrocinados:** Son mensajes privados de LinkedIn que se envían a los usuarios directamente a sus buzones. Son exactamente igual que los mensajes normales y corrientes, solo que llevan el distintivo de "patrocinado" y una pequeña llamada a la acción.

2.5.5 Spotify

El mundo de la música ya no viene en forma de un disco o de cassette. Como ya se sabe, la música se ha vuelto digital, al igual que con todo lo que solíamos usar de forma tangible, como libros, imágenes y periódicos.

Uno de los servicios más populares en el mercado de la música digital es Spotify, un servicio de transmisión de música comercial que se lanzó en octubre de 2008 por la empresa sueca Spotify AB y cuenta con más de 150 millones de usuarios. Spotify ofrece una amplia variedad de música a sus oyentes y suscriptores, ya que se han asociado con varios sellos discográficos que incluyen Sony, EMI, Warner Music Group y Universal.

Spotify debe ser considerado una red de medios sociales. ¿Por qué? Spotify, con su integración con Facebook y Twitter, ha permitido a los oyentes aprovechar sus seguidores en esas redes sociales, creando al mismo tiempo una plataforma social propia. En este sentido, al ofrecer otra forma de conectar y socializar con amigos, Spotify se ha convertido en la red social más nueva.

Spotify no solo permite que sus usuarios se conecten con la música que proporcionan, sino que también les ayuda a conectarse con sus amigos. Spotify permite a los usuarios integrar su cuenta con las cuentas de Facebook y Twitter

existentes. A partir de entonces, podrán acceder a la música y las listas de reproducción favoritas de sus amigos/seguidores.

La compatibilidad con Facebook permite a los usuarios de Spotify compartir música con amigos de Facebook mediante el uso de la bandeja de entrada del servicio. Los usuarios de Spotify pueden enviar pistas o listas de reproducción a amigos que luego pueden reproducirlas a través de su cuenta de Spotify.

Spotify está aprovechando el hecho de que la música es un elemento que une a las personas, y que las relaciones con la música son profundamente personales e íntimas. La música es una forma de autoexpresión; dando forma a las distintas identidades. La integración de Spotify con las plataformas de medios sociales nos permite resaltar esas expresiones.

Spotify ha cambiado la dinámica entre los compromisos personales y sociales con la música. Anteriormente, cuando escuchábamos música, éramos solo nosotros quienes sabíamos qué canción estábamos escuchando en ese momento. Con la integración de Spotify con las plataformas de redes sociales como Twitter y Facebook, esas elecciones o preferencias en la selección de música pueden tomarse como sugerencias o recomendaciones para nuestros amigos; por lo tanto, inspiran inadvertidamente a nuestros amigos y los hábitos de escuchar música que suceden más adelante.

A medida que más usuarios continúan escuchando Spotify y compartiendo su consumo de música con sus amigos, prevemos que el propósito de la aplicación cambie de un lugar para escuchar música a uno para compartirla.

Como un canal de redes sociales, Spotify parece ser un nuevo canal para capitalizar y generar conciencia (y clientes potenciales) hacia una empresa o marca.

Spotify Ad Studio es una nueva plataforma de publicidad de autoservicio donde es posible administrar y crear fácilmente campañas publicitarias. Con Ad Studio, se puede:

- Crear campañas publicitarias de imagen, audio o video.
- Estimar el alcance de las mismas basadas en el público objetivo.
- Administrar todo tipo de informes de campaña.

Los anuncios de audio o video se sirven a los oyentes de suscripción gratuita de Spotify (alrededor de 101 millones de ellos) durante los descansos de las canciones. Es posible dirigir el anuncio por género, ubicación, género, edad, actividad y dispositivo. También se cuenta con la propiedad del área Cover Art, lo que permite hacer clic en el anuncio y dirigir el tráfico a un destino de URL.

Recientemente, las marcas han estado construyendo listas de reproducción y perfiles personalizados para impulsar los compromisos con su mercado objetivo. Carnival Cruise Lines creó la lista de reproducción Carnival Cruise Tunes para involucrar a las personas dentro y fuera del barco. Con música inspiradora y tropical, esta lista de reproducción hace que los oyentes no solo se sientan como si estuvieran en un crucero, sino que también se sientan como si estuvieran en uno. Esta lista de reproducción no termina solo en Spotify, los usuarios pueden compartir la música que están escuchando en otras formas de redes sociales, solo extendiendo el alcance de esta lista de reproducción.

Es importante tener en cuenta que estas listas de reproducción deben tratarse como anuncios. La selección y el orden de las canciones deben ser altamente considerados. Esta forma de publicidad hace que su marca se sienta genuina y realista, por lo que es importante que sus esfuerzos sean paralelos a eso.

Starbucks es otro gran ejemplo de una marca que utiliza Spotify. Se asociaron en 2015 porque Spotify necesitaba suscriptores y Starbucks necesitaba listas de reproducción confiables. Spotify ahora está vinculado a la aplicación Starbucks, donde puede conectarse fácilmente a la música que se está reproduciendo actualmente. Su perfil de Spotify tiene casi 85.000 seguidores con listas de reproducción actualizadas casi todos los días.

En síntesis, Spotify está anclado a un elemento tan personal como es el gusto musical, y eso lo convierte en una potencial red social. Con el auge del social media, era natural que los usuarios comenzaran a crear comunidad en torno a la música y pasará de ser un simple servicio a convertirse en un conector entre artistas y fans.

Sin embargo, Spotify por sí sola no se sostiene, aún, como red social. Tal y como está planteada la app, necesita el apoyo de otras redes como Facebook para que la interacción sea 100% real.

CAPÍTULO 3. Descripción del problema

IMS es una empresa de Marketing Digital y una de las líderes en su segmento en América Latina. Su objetivo es llevar los mensajes de las marcas a las audiencias correctas.

Hoy en día, el proceso de carga de una campaña digital consta de demasiadas tareas humanas; ésta tarea es realizada por los Accounts Managers (a partir de ahora AM), quienes son los representantes y responsables de las distintas redes sociales (a partir de ahora plataformas) con las cuales trabaja IMS. El flujo comienza con un cliente que se comunica con un AM de una plataforma en particular, solicitando la creación de una campaña. El AM es el encargado de recolectar toda la información de la campaña (fecha de lanzamiento, monto, formatos/productos, creativos), guiar a los clientes en cómo segmentar su campaña, para llegar a la audiencia deseada y lograr un buen rendimiento, y gestionar todos los permisos y particularidades de la plataforma para que el cliente pueda disponer de su campaña en la plataforma solicitada.

Una vez que el AM cuenta con toda la información necesaria, es él quien carga a través de un proceso manual la campaña en la plataforma indicada. Todas las tareas llevadas a cabo por el AM, son volcadas en Salesforce, un CRM[6] interno donde registran cada una de las ventas/oportunidades. Es aquí donde llevan registro de los distintos estadíos por los cuales pasó una campaña, desde ser una posible oportunidad hasta consolidarse como una venta concreta.

La principal falencia que presenta este modelo de trabajo es que si un cliente desea publicitar en más de una plataforma, debe comunicarse con los distintos AM de IMS y comunicarle a cada uno lo que desea hacer.

El desafío de esta tesina radica en crear una solución de software para que IMS pueda minimizar el trabajo interno a la hora de brindar el servicio de publicación de campañas en redes sociales, permitiéndoles a los AM dedicarse exclusivamente a los clientes más importantes/redituables, y que dicha solución

[6] Customer Relationship Management. Es un término que se usa en el ámbito del marketing y ventas. Traducido al castellano significa gestión de relaciones con clientes.

se integre con otras herramientas internas de IMS para dar soporte al resto del negocio.

En este contexto se propone el sistema ONE como solución, una plataforma que permite a las agencias y/o anunciantes autogestionar sus campañas publicitarias en distintas plataformas digitales desde un único lugar, ofreciendo una experiencia similar o superior a la que brinda la plataforma en sí.

Además de mejorar la gestión de carga de campañas ONE incorpora inteligencia de negocio que permite:

- Segmentar las campañas basadas en recomendaciones internas, llamadas Clusters.
- Segmentar las campañas libremente con la posibilidad guardar esa audiencia para poder aplicarla en otras campañas.
- Realizar recomendaciones de plataformas y productos basados en los objetivos de la campaña.

CAPÍTULO 4. Análisis de tecnologías

En este capítulo se analizarán las tecnologías necesarias para desarrollar la solución a los problemas descritos anteriormente. La misma deberá permitir la creación de campañas publicitarias en múltiples plataformas por parte del cliente, permitiéndole seleccionar los productos, creativos y segmentar cada plataforma de la forma que desee para poder lograr el objetivo y llegar a la audiencia deseada.

El sistema permitirá crear campañas en Twitter, LinkedIn y Spotify. Para Twitter y LinkedIn la publicación de las campañas será de manera automática ya que se integrarán sus respectivas APIs en ONE, reduciendo así la interacción de los AM de IMS.

Por otro lado, la solución deberá amoldarse a las reglas de negocio que presenta IMS, pudiendo presentarse algunas como restricciones para ciertas acciones del usuario y deberá integrarse con las distintas herramientas que hoy en día maneja IMS para manejar ventas y generar reportes.

Por último, la solución deberá modelarse de manera que resulte flexible y adaptable ante la incorporación de nuevas plataformas, garantizando que el impacto sobre lo que esté desarrollado sea mínimo.

4.1 JavaScript

Desde su lanzamiento en 1995, JavaScript ha pasado por muchos cambios. Al principio, hizo mucho más sencillo agregar elementos interactivos a las páginas web. Luego se volvió más robusto con DHTML y AJAX. Ahora, con Node.js, JavaScript se ha convertido en un lenguaje que se utiliza para crear aplicaciones tanto del lado cliente como del lado servidor. El comité que es y ha estado a cargo de llevar a cabo los cambios a JavaScript es la Asociación Europea de Fabricantes de Computadoras (ECMA).

JavaScript es un lenguaje de programación interpretado de alto nivel que también se caracteriza por ser dinámico, débilmente tipado, basado en prototipos y multi-paradigma.

Junto con HTML y CSS, es una de las tres tecnologías principales de la Web. La gran mayoría de los sitios web lo utilizan y todos los principales navegadores web tienen un motor de JavaScript dedicado para ejecutarlo.

Como lenguaje de paradigma múltiple, JavaScript es compatible con los estilos de programación controlados por eventos, funcionales e imperativos (incluidos los orientados a objetos y los prototipos). Tiene API para trabajar con texto, matrices, fechas, expresiones regulares y el DOM, pero el lenguaje en sí no incluye ninguna Entrada/Salida, como redes, almacenamiento o instalaciones gráficas, que se basan en el entorno host en el que está incrustado.

Inicialmente fue solo implementado en el lado del cliente en los navegadores web, ahora los motores de JavaScript están integrados en muchos otros tipos de software de host, incluido el lado del servidor en servidores web y bases de datos, y en programas no web tales como procesadores de texto y software PDF, y en tiempo de ejecución entornos que hacen que JavaScript esté disponible para escribir aplicaciones móviles y de escritorio, incluidos widgets de escritorio. (Brown, 2016).

4.2 Programación funcional con JavaScript

La tendencia funcional proviene de la década de 1930, con la invención del cálculo lambda o λ-calculus. Las funciones han sido parte del cálculo desde que surgió en el siglo XVII. Las funciones se pueden enviar a las funciones como argumentos o se pueden devolver desde las funciones como resultados. Las funciones más complejas, llamadas funciones de orden superior, pueden manipular las funciones y usarlas como argumentos o resultados o como ambos. En la década de 1930, Alonzo Church estaba en Princeton experimentando con estas funciones de orden superior cuando inventó el cálculo lambda.

A fines de la década de 1950, John McCarthy tomó los conceptos derivados de λ-calculus y los aplicó a un nuevo lenguaje de programación llamado Lisp. Lisp implementó el concepto de funciones de orden superior y como miembros de primera clase. Una función se considera un miembro de primera clase cuando

puede declararse como una variable y enviarse a funciones como un argumento. Estas funciones pueden incluso ser devueltas desde funciones.

JavaScript es compatible con la programación funcional porque las funciones de JavaScript son de primera clase. Esto significa que las funciones pueden hacer las mismas cosas que las variables.

En Junio del 2015, se aprobó la especificación de JavaScript llamada ECMAScript 6 ó ES6. Ésta especificación agrega mejoras de lenguaje que refuerzan las técnicas de programación funcional, destacándose entre ellas:

- Arrow functions
- Promises
- Spread Operator

4.2.1 Arrow functions

La expresión de *arrow function* tiene una sintaxis más corta que una expresión de función convencional y no vincula sus propios *this*, *arguments*, *super*, o *new.target*. Las *arrow functions* siempre son anónimas, no están relacionadas con métodos y no pueden ser usadas como constructores.

Antes de las *arrow functions*, cada nueva función definía su propio valor de *this*, lo cual resultaba molesto cuando se intentaba aplicar programación orientada a objetos.

En ECMAScript 3/5, este problema fue corregido asignándole el valor de *this* a una variable por fuera de la función interna. Una *arrow function* no tiene su propio this; utiliza el valor del contexto de ejecución que la contiene.

Con las arrow functions, se pueden crear funciones sin utilizar la palabra clave de *function*, así como también es posible no tener que usar la palabra clave return.

```
var lordify = function(firstname) {
  return `${firstname} of Canterbury`
}

console.log( lordify("Dale") )       // Dale of Canterbury
console.log( lordify("Daryle") )     // Daryle of Canterbury
```

Fig. 3: Función tradicional.

La Fig. 4 muestra la funcionalidad anterior utilizando arrow function. Puede observarse que la codificación luce más sencilla.

```
var lordify = firstname => `${firstname} of Canterbury`
```

Fig. 4: Arrow function.

En síntesis:

- Con *arrow function*, es posible declarar una función completa en una línea.
- Se elimina la palabra clave de *function*.
- Se elimina el retorno porque la flecha apunta a lo que debe devolverse.
- Si la función solo toma un argumento, es posible eliminar los paréntesis de los argumentos.

4.2.2 Promises

Una promesa representa un valor que puede estar disponible ahora, en el futuro, o nunca.

Una Promesa es un proxy para un valor no necesariamente conocido en el momento que es creada la promesa. Permite asociar manejadores que actuarán asincrónicamente sobre un eventual valor en caso de éxito, o la razón de falla en caso de una falla. Esto permite que métodos asíncronos devuelvan valores como si fueran síncronos: en vez de inmediatamente retornar el valor final, el método asíncrono devuelve una promesa de suministrar el valor en algún momento en el futuro.

Una Promesa se encuentra en uno de los siguientes estados:

- Pendiente (pending): estado inicial, no cumplida o rechazada.
- Cumplida (fulfilled): significa que la operación se completó satisfactoriamente.
- Rechazada (rejected): significa que la operación falló.

Una promesa pendiente puede ser cumplida con un valor, o rechazada con una razón (error). Cuando cualquiera de estas dos opciones sucede, los métodos asociados, encolados por el método then de la promesa, son llamados. Si la

promesa ya ha sido cumplida o rechazada en el momento que es anexado su correspondiente manejador, el manejador será llamado, de tal manera que no exista una condición de carrera entre la operación asíncrona siendo completada y los manejadores siendo anexados.

Como los métodos *Promise.prototype.then()* y *Promise.prototype.catch()* retornan promesas, éstas pueden ser encadenadas.

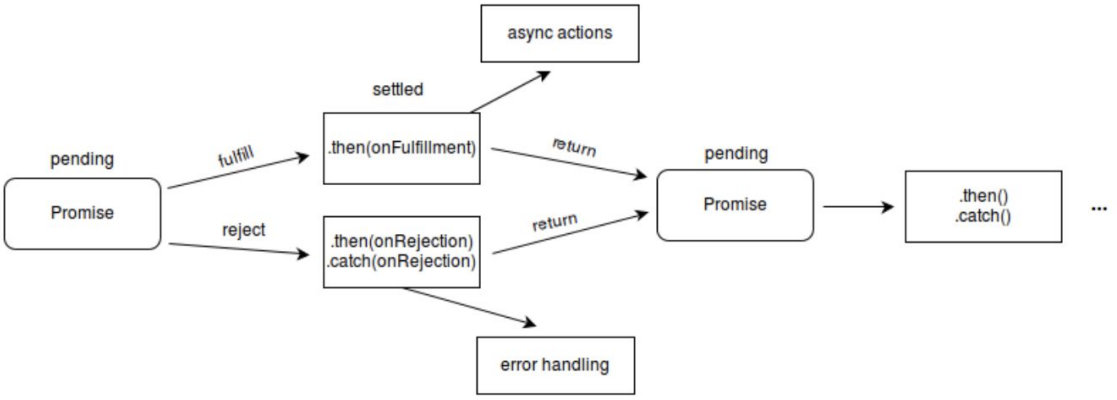

Fig. 5: Encadenamiento de promesas.

Las promesas hacen que el manejo de solicitudes asíncronas sea más fácil, lo cual es bueno, ya que se tiene que lidiar con una gran cantidad de datos asíncronos en JavaScript.

4.2.3 Spread Operator

El operador de propagación ó spread operator permite que una expresión sea expandida en situaciones donde se esperan múltiples argumentos (llamadas a funciones) o múltiples elementos (arrays literales).

El operador de propagación se indica con tres puntos (...), los cuales realizan varias tareas diferentes.

Primero, el operador de propagación nos permite combinar los contenidos de los arreglos. Por ejemplo, si se tienen dos arreglos, se puede hacer un tercero que combine las dos primeros en él:

```
var peaks = ["Tallac", "Ralston", "Rose"]
var canyons = ["Ward", "Blackwood"]
var tahoe = [...peaks, ...canyons]

console.log(tahoe.join(', '))  // Tallac, Ralston, Rose, Ward, Blackwood
```

Fig. 6: Combinación de arreglos con Spread Operator.

También se puede utilizar el operador de propagación para recopilar argumentos de función como una arreglo.

```
function directions(...args) {
  var [start, ...remaining] = args
  var [finish, ...stops] = remaining.reverse()

  console.log(`drive through ${args.length} towns`)
  console.log(`start in ${start}`)
  console.log(`the destination is ${finish}`)
  console.log(`stopping ${stops.length} times in between`)
}

directions(
    "Truckee",
    "Tahoe City",
    "Sunnyside",
    "Homewood",
    "Tahoma"
)
```

Fig. 7: Recopilación de argumentos con Spread Operator.

El operador de propagación también se puede usar para objetos. Usar el operador de propagación con objetos es similar a usarlo con arreglos.

```
var morning = {
  breakfast: "oatmeal",
  lunch: "peanut butter and jelly"
}

var dinner = "mac and cheese"

var backpackingMeals = {
  ...morning,
  dinner
}

console.log(backpackingMeals) // {breakfast: "oatmeal",
                                  lunch: "peanut butter and jelly",
                                  dinner: "mac and cheese"}
```

Fig. 8: Combinación de objetos con Spread Operator.

4.2.4 Imperativo vs. declarativo

La programación funcional es parte de un paradigma de programación más amplio: la programación declarativa. La programación declarativa es un estilo de programación en el que las aplicaciones se estructuran de una manera que prioriza la descripción de lo que debería suceder en vez de definir cómo debería suceder.

Para entender la programación declarativa, se comparará con la programación imperativa, o un estilo de programación que solo concierne a cómo lograr resultados con código. Consideremos una tarea común: hacer que una cadena sea compatible con la URL. Por lo general, esto se puede lograr reemplazando todos los espacios de una cadena por guiones, ya que los espacios no son compatibles con URL. Primero, examinemos un enfoque imperativo para esta tarea:

```
var string = "This is the midday show with Cheryl Waters";
var urlFriendly = "";

for (var i=0; i<string.length; i++) {
  if (string[i] === " ") {
    urlFriendly += "-";
  } else {
    urlFriendly += string[i];
  }
}

console.log(urlFriendly);
```

Fig. 9: Ejemplo imperativo.

En este ejemplo, se recorre cada carácter de la cadena, reemplazando los espacios a medida que ocurren. La estructura de este programa solo se ocupa de cómo se puede lograr dicha tarea. Utiliza un bucle for y una declaración if, y establece valores con un operador de igualdad. Mirando únicamente el código no dice mucho. Los programas imperativos requieren muchos comentarios para comprender lo que está sucediendo.

Ahora veamos un enfoque declarativo del mismo problema:

```
const string = "This is the mid day show with Cheryl Waters"
const urlFriendly = string.replace(/ /g, "-")

console.log(urlFriendly)
```

Fig. 10: Ejemplo declarativo.

Aquí se utiliza *string.replace* junto con una expresión regular para reemplazar todas las instancias de espacios con guiones. Usar *string.replace* es una forma de describir lo que se supone que debe suceder: los espacios en la cadena deben reemplazarse. Los detalles de cómo se tratan los espacios se resumen en la función de reemplazo. En un programa declarativo, la sintaxis en sí misma describe lo que debería suceder y los detalles de cómo suceden las cosas se abstraen.

Esencialmente, la programación declarativa produce aplicaciones que son más fáciles de razonar y, cuando es más fácil razonar acerca de una aplicación, dicha aplicación es más fácil de escalar.

4.3 Node.js

Node.js es una plataforma construida sobre la máquina virtual de JavaScript V8, el cual es el motor de JavaScript de alto rendimiento, bajo consumo de memoria y de código abierto de Google, escrito en C++ que implementa ECMAScript y WebAssembly. V8 se puede ejecutar de forma independiente o se puede incrustar en cualquier aplicación de C++. Fue implementada para crear aplicaciones de red de manera rápida y escalables. Implementa un modelo de entrada/salida no bloqueante dirigido por eventos que lo hace ligero y eficiente, ideal para aplicaciones de tiempo real que manejan mucha intensidad de datos entre dispositivos distribuidos.

A diferencia de otros entornos modernos, los procesos de Node.js no se basan en multithreading (múltiples hilos) para dar soporte a la programación concurrente, sino que se basa en un modelo de eventos asíncronos de Entrada/Salida. Se puede pensar en un proceso servidor de Node.js como un demonio

single-threaded (único hilo) que tiene embebido el motor JavaScript V8 para soportar customización del mismo (Teixeira, 2013).

Para esta plataforma, el lenguaje de programación JavaScript es un complemento excelente ya que soporta eventos callbacks, es decir, permite crear funciones anónimas que se pueden registrar como gestoras o controladoras (handler) de eventos para ser ejecutadas cuando los mismos ocurran.

Node.js, al ser un entorno JavaScript en el lado del servidor, brinda algunos beneficios extras:

- Los desarrolladores pueden escribir aplicaciones web en un solo lenguaje de programación, lo que ayuda a reducir el cambio de contexto entre el desarrollo del cliente y el del servidor, y permite reutilizar código entre ambos.
- JSON (JavaScript Object Notation) es un formato de intercambio de datos actualmente muy utilizado y su sintaxis deriva de JavaScript.
- JavaScript es un lenguaje utilizado en varias bases de datos NoSQL (Por ejemplo, MongoDB), por lo tanto reduce la complejidad en la integración con las mismas.
- Utiliza la máquina virtual V8 que implementa el estándar ECMAScript, es decir, se pueden aprovechar las nuevas características del lenguaje JavaScript directamente en Node, sin necesidad de esperar a que los navegadores las implementen.

4.3.1 Bloques de construcción en Node

Powers (2016) sostiene que, aunque ambos están basados en JavaScript, los entornos entre las aplicaciones basadas en navegador y las aplicaciones Node.js son muy diferentes. Una diferencia fundamental entre Node y JavaScript basado en navegador es el generador de datos binarios.

Es cierto que Node ahora tiene acceso al ArrayBuffer ES6 y a los arreglos tipados. Sin embargo, la mayoría de las funciones de datos binarios en Node se implementan con la clase Buffer.

Node posee dos objetos fundamentales, *global* y *process.* El objeto *global* es algo similar al objeto *global* en el navegador, con algunas diferencias muy importantes. El objeto *process*, sin embargo, es puramente de Node.

4.3.1.1 Objeto *global*

En el navegador, cuando se declara una variable en el nivel superior, se declara globalmente. En cambio, cuando se declara una variable en un módulo o aplicación en Node, la variable no está disponible globalmente; está restringida al módulo o a la aplicación. Por lo tanto, se puede declarar una variable "global" llamada *str* en un módulo y también en la aplicación que usa el módulo, y no habrá ningún conflicto.

El objeto *global* comparte globalmente en todos los entornos el acceso a todos los objetos y funciones de Node disponibles globalmente.

4.3.1.2 Objeto *process*

El objeto *process* es un componente esencial ya que proporciona acceso a la información sobre el entorno Node, así como el entorno de ejecución. Además, la entrada/salida estándar ocurre a través de *process*, así como también con él es posible terminar con una aplicación de Node, e incluso notificar cuando el bucle de eventos de Node ha finalizado un ciclo.

4.3.2 Modelo orientado a eventos vs. Modelo multithreading

En la programación tradicional se ejecutan las operaciones de Entrada/Salida de la misma manera que se ejecutan las llamadas a funciones locales, es decir, el procesamiento no continua hasta que las operaciones o llamadas hayan finalizado. Este modelo de programación bloqueante proviene desde los tiempos de los sistemas “time-sharing” en los cuales cada proceso se correspondía con una tarea humana. En esos sistemas, los usuarios típicamente necesitaban terminar una tarea antes de decidir la siguiente tarea a realizar. Este modelo basado en un proceso por usuario no escaló muy bien ya que gestionar varios procesos le genera una gran carga al sistema operativo, ya sea en memoria como

así también por los costos producidos por los cambios de contexto de los mismos, y la performance de las tareas tendían a decaer luego de alcanzar una cantidad determinada.

De este modelo surge una alternativa, la programación Multithreading o multi-hilos. Un thread o hilo es, básicamente, un proceso liviano que comparte memoria con los demás threads dentro del mismo proceso. Mientras un thread está esperando a que finalice una operación de entrada/salida, otro thread puede asumir el control de la CPU (Unidad Central de Procesamiento) para un uso más óptimo de la misma. Algunos sistemas permiten que los threads se ejecuten en paralelo en diferentes núcleos de CPU, lo cual requiere el uso de primitivas de sincronización para el acceso concurrente a datos en memoria compartida.

Por ejemplo, aplicaciones como servidores Web llevan a cabo una cantidad significante de operaciones de entrada/salida, por lo tanto, el uso de múltiples threads permitiría a este tipo de aplicaciones hacer un mejor uso de los procesadores. En un sistema moderno de varios núcleos (multi-core) se pueden ejecutar threads simultáneamente en diferentes núcleos, es decir, con paralelismo real. En un sistema con un único núcleo, el procesador ejecuta un thread, cambia a otro thread y lo ejecuta, y así continuamente. Por ejemplo, el procesador cambia su contexto de ejecución para otro thread cuando el thread actual necesita escribir en un socket TCP. El cambio en el contexto de ejecución ocurre debido a que completar la operación de escritura podría usar varios ciclos del procesador y, en lugar de desperdiciar ciclos de procesamiento esperando que la operación de escritura termine, el procesador inicia la operación de entrada/salida y cambia su contexto de ejecución para ejecutar otro thread, manteniéndose ocupado haciendo trabajo útil. Cuando la operación de entrada/salida finaliza el thread está nuevamente listo para ser ejecutado.

La programación orientada a eventos, también llamada programación asíncrona, es un estilo de programación donde el flujo de ejecución es determinado por eventos. Los eventos son manejados por gestores de eventos (event handlers) o funciones callbacks. Una función callback es invocada cuando un evento significante ocurre, como por ejemplo, cuando se encuentra disponible el

resultado de una consulta a la base de datos o cuando un usuario hace clic en un botón.

Las operaciones de entrada/salida asíncronas son importantes para el modelo de programación orientada a eventos porque evitan que las aplicaciones se bloqueen mientras esperan que termine la ejecución de la operación. Por ejemplo, si una aplicación llena el buffer de un socket al estar escribiendo en el mismo y la operación de entrada/salida es bloqueante, el socket mantendría a la aplicación en espera hasta que haya lugar disponible para escritura en el buffer impidiéndole realizar un trabajo más útil. En cambio, si la operación de entrada/salida es no bloqueante, el socket retornaría un mensaje indicando que por el momento no se puede seguir escribiendo en el buffer y notificará posteriormente a quien corresponda cuando haya disponibilidad para escritura. Asumiendo que la aplicación se suscribe como interesada en este tipo de evento del socket, puede continuar realizando otras tareas sabiendo que recibirá la notificación de un evento cuando el buffer de escritura del socket tenga espacio disponible.

4.3.3 Fundamentos de programación en Node.js

El estilo de programación orientado a eventos de Node.js es una de las características que lo define. Este modelo está acompañado por un bucle de eventos (event loop). Un bucle de eventos es un estructura que principalmente lleva a cabo dos funciones en un bucle continuo, detección de eventos y propagación de eventos para su gestión. Durante su ejecución, el bucle de eventos debe detectar qué eventos ocurren y ante la ocurrencia de los mismos, debe determinar la función callback correspondiente e invocarla.

Tilkov y Vinoski (2010) concuerdan en que el bucle de eventos se puede definir como un thread ejecutándose dentro de un proceso, lo que significa que cuando un evento ocurre, el controlador (handler) del evento puede ejecutarse sin interrupciones. Es decir, dado un momento determinado a lo sumo sólo habrá ejecutándose un controlador de eventos y cualquiera de estos en ejecución podrá completar su tarea sin ser interrumpido. Esto le permite a los programadores abstraerse de los requisitos necesarios para modelar la sincronización entre

threads y no tener que preocuparse por el estado de la memoria compartida y la ejecución concurrente de los threads.

Node.js brinda la posibilidad de utilizar JavaScript del lado del servidor, de la misma manera que los navegadores brindan la posibilidad de utilizar JavaScript del lado del cliente. Por este motivo, es importante entender cómo funcionan los navegadores con el fin de entender cómo funciona Node.js. Ambos implementan un bucle de eventos y son no bloqueantes para realizar operaciones de entrada/salida.

Cuando es necesario llevar a cabo una operación de entrada/salida en el navegador, la misma se realiza fuera del bucle de eventos y cuando la misma finaliza se emite un evento que es manejado por la función callback correspondiente.

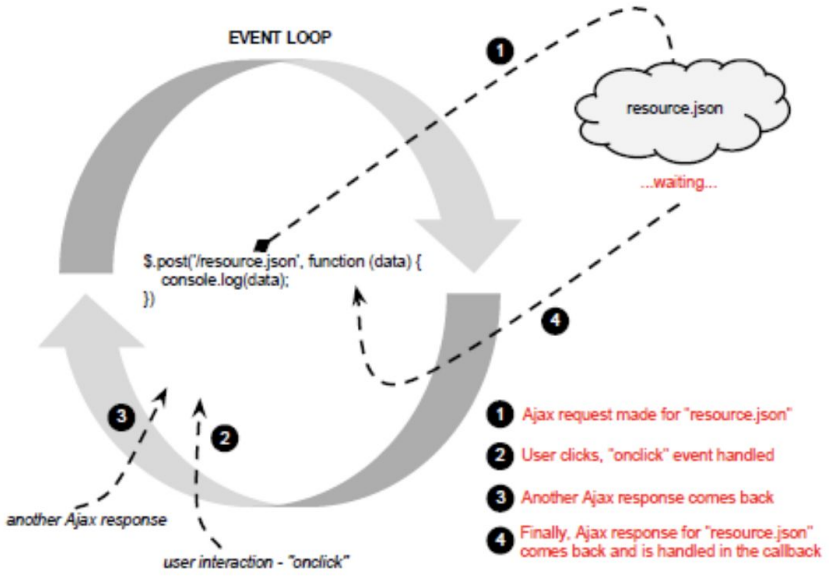

Fig. 11: Operación de entrada/salida no bloqueante en un navegador web.

La operación de entrada/salida del ejemplo es llevada a cabo de manera asíncrona y no bloqueante respecto al principal hilo de ejecución. De esta manera, el bucle de eventos puede atender cualquier otra interacción o solicitud que provenga desde la página. Esto le permite al navegador poder responder a las necesidades del usuario y gestionar gran cantidad de interacciones del mismo con la página sin espera innecesaria.

En Node.js, las operaciones de entrada/salida casi siempre son llevadas a cabo fuera del bucle principal de eventos (main event loop), permitiendo al servidor recibir solicitudes y conexiones eficiente y constantemente.

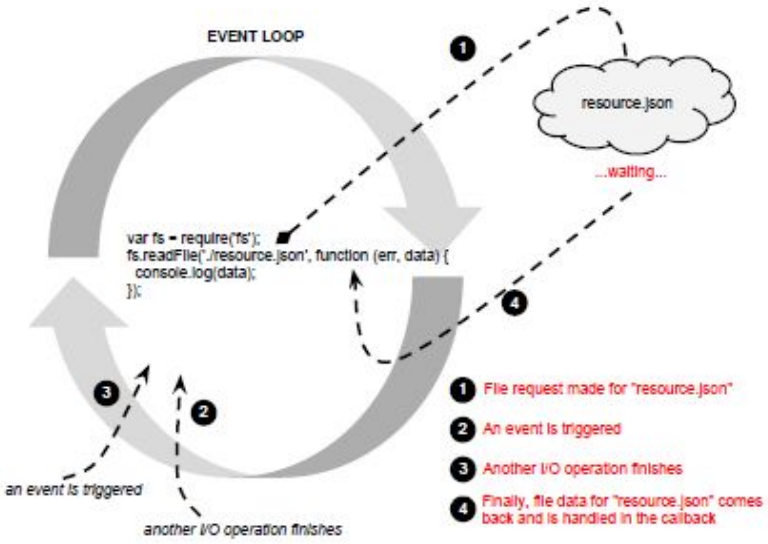

Fig. 12: Operación de entrada/salida no bloqueante en Node.js.

En el ejemplo anterior, se procesa una solicitud de lectura del archivo “resource.json” (1) de manera asíncrona y se prosigue con la ejecución del script sin esperar que la operación de entrada/salida finalice. En segundo lugar se dispara un evento (2), luego finaliza una operación de entrada/salida diferente (3) y por último (4) se recuperan y se imprimen en consola los datos del archivo solicitados en el paso (1).

Al gestionar los eventos de manera asíncrona, rara vez permite que un proceso alcance un tiempo límite de entrada/salida ya que la latencia producida no sería lo suficientemente grande como para bloquear el servidor, como si puede pasar en las operaciones de entrada/salida bloqueantes. Este modelo le permite a Node.js ser un servidor liviano y/o ligero para llevar a cabo las operaciones que, generalmente, un servidor que maneja comunicaciones sincrónicas las realiza de manera lenta.

4.3.4 Módulos

Node.js implementa el estándar CommonJS para modularización. Cada módulo tiene su propio contexto, lo que significa que no afectan el contexto global de la aplicación ni interfieren con otros módulos.

En Node.js, los módulos están referenciados por la ruta del archivo o por nombre. Aquellos que vienen integrados en Node.js son cargados cuando el proceso Node.js arranca. Además de los del núcleo de Node.js, existe una amplia cantidad de módulos desarrollados por terceros que se pueden instalar usando el gestor de dependencias de Node.js llamado NPM (Node Package Manager) o se pueden importar módulos desarrollados por uno mismo. (Cantelon, Harter, Holowaychuk, & Rajlich, 2013).

Cualquier módulo de cualquiera de los tipos mencionados anteriormente, expone una API pública que el programador puede utilizar, luego de que el módulo haya sido importado o requerido en el script correspondiente mediante la función *require*.

El sistema de módulos CommonJS es la única manera de compartir objetos o funciones entre archivos en Node. Para definir lo que se quiere exponer públicamente en cada módulo se utiliza el objeto export del módulo actual. Para hacer referencia al módulo en el que se está situado, existe la variable module, por lo tanto, con module.exports se exporta la funcionalidad deseada para los scripts que importen ese módulo.

Node.js trae incorporado varios módulos compilados dentro de su distribución. Estos son llamados módulos del núcleo, y su importación se hace por su nombre y no por la ruta del archivo, y son preferentemente cargados incluso si existen módulos de terceros con el mismo nombre. Para importar módulos instalados mediante NPM se debe proveer de la ruta de archivo, ya sea absoluta o relativa al archivo actual. En el caso de que se indique el nombre del módulo solamente y el mismo no sea del núcleo, Node.js intentará buscarlo dentro de la carpeta *node_modules* en el directorio actual. Si falla la búsqueda del archivo, intentará

localizar la carpeta *node_modules* del directorio padre y así continuamente hasta alcanzar el directorio raíz o encontrar el módulo que se está buscando.

La carpeta mencionada anteriormente *node_modules* es el directorio por defecto donde NPM descarga los módulos que van a ser usados en Node.js.

4.3.5 LoopBack

LoopBack es el framework con el cual se desarrolló el backend de ONE, está constituido por un conjunto de módulos Node.js los cuales se pueden utilizar de forma independiente o en conjunto para crear una API REST.

Una aplicación LoopBack interactúa con las fuentes de datos a través del modelo de API de LoopBack, disponible localmente en Node.js, de forma remota a través de REST y a través de las API de cliente nativas para iOS, Android y HTML5. Usando estas API, las aplicaciones pueden consultar bases de datos, almacenar datos, cargar archivos, enviar correos electrónicos, crear notificaciones automáticas, registrar usuarios y realizar otras acciones proporcionadas por las fuentes de datos y los servicios.

Los clientes pueden llamar a las API de LoopBack directamente usando Strong Remoting, una capa de transporte conectable, que permite proporcionar una API de backend a través de REST, WebSockets y otros transportes.

El siguiente diagrama ilustra los módulos principales de LoopBack, cómo están relacionados y sus dependencias.

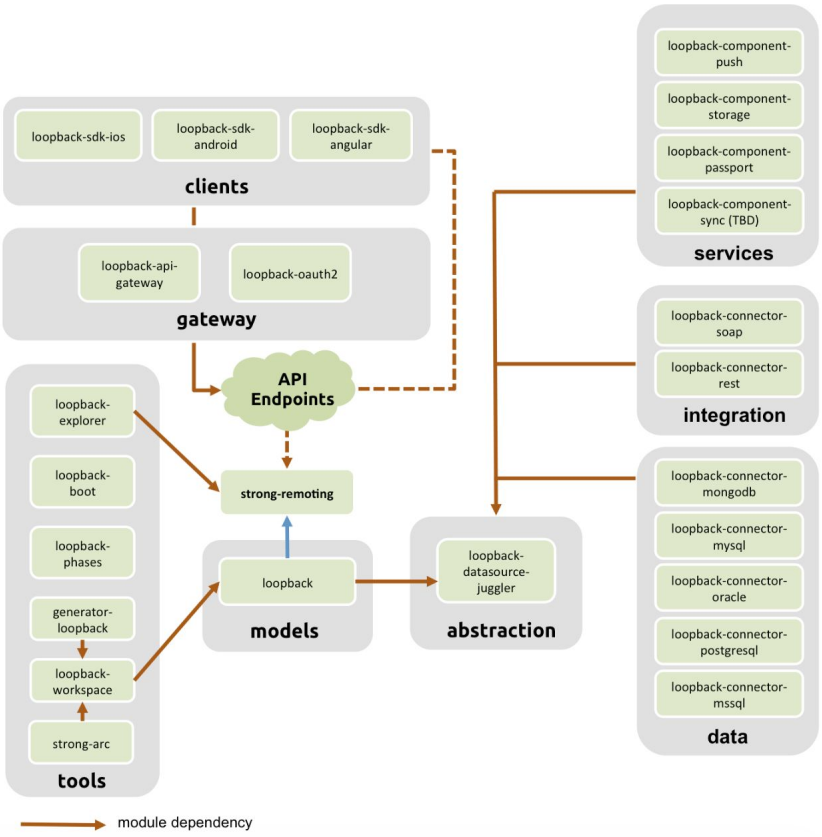

Fig. 13: Módulos principales de LoopBack.

4.3.5.1 Modelos

Los modelos son parte del corazón de LoopBack y representan fuentes de datos como bases de datos u otros servicios backend (REST, SOAP, etc.). Los modelos LoopBack son objetos de JavaScript con API de REST y Node.

Una característica clave de LoopBack es que cuando define un modelo, viene automáticamente con una API REST predefinida con un conjunto completo de operaciones de creación, lectura, actualización y eliminación.

El objeto del Modelo base tiene métodos para agregar hooks y validar datos. Todos los objetos modelo heredan de él. Los modelos tienen una jerarquía de herencia, como se muestra en la Fig. 14: cuando adjunta un modelo a un origen de datos persistente, se convierte en un modelo conectado con las operaciones de crear, recuperar, actualizar y eliminar. Los modelos incorporados de LoopBack heredan de él.

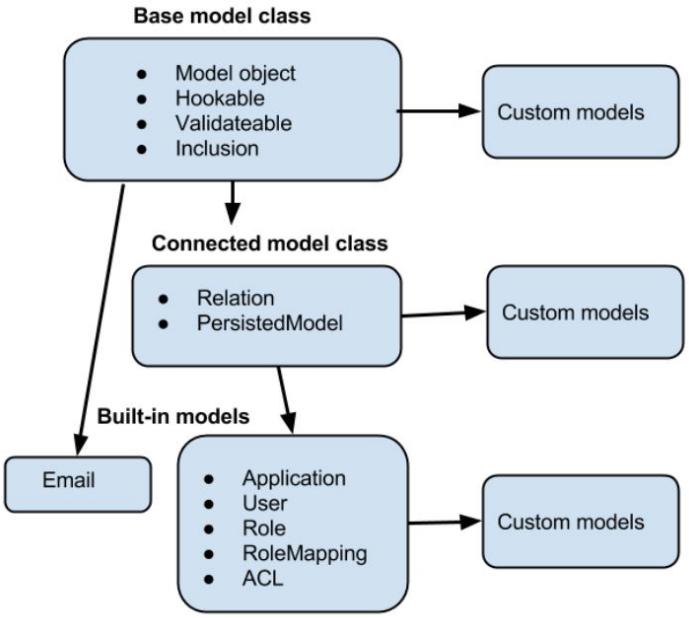

Fig. 14: Modelos principales de LoopBack.

LoopBack tiene un conjunto de modelos predefinidos incorporados, como Usuario, Rol y Aplicación.

LoopBack generaliza servicios de backend como bases de datos, API REST, servicios web SOAP y servicios de almacenamiento como fuentes de datos.

Las fuentes de datos están respaldadas por conectores que luego se comunican directamente con la base de datos u otro servicio de backend. Las aplicaciones no usan conectores directamente, sino que pasan por las fuentes de datos utilizando las API DataSource y PersistedModel.

4.3.5.2 Enrutamiento

LoopBack se basa en Express e implementa el sistema de enrutamiento de Express. Sin embargo, el enrutamiento básico de Express está limitado solo a una pequeña parte de la funcionalidad de LoopBack. Una gran parte de las características de LoopBack se implementan utilizando una extensión más detallada del sistema de enrutamiento.

Algunos de los puntos claves del enrutamiento son:

- El enrutamiento se refiere a las reglas para capturar solicitudes al servidor, y el posterior paso y manejo de solicitudes a través de una cadena de funciones de middleware.

- Una función de middleware acepta tres objetos, el objeto de solicitud (req), el objeto de respuesta (res) y el siguiente middleware en la cadena (next); en ese orden.
- El middleware se carga utilizando *app.use()* o asignándolo como la función de devolución de llamada de una definición de ruta.
- Múltiples middleware pueden manejar las solicitudes a una ruta, estos middleware coincidentes conforman la cadena de middleware para la solicitud. La solicitud pasará a través de cada middleware en el orden en que se cargaron, a menos que uno de los middleware en la cadena termine la propagación.
- Cualquier middleware en la cadena puede terminar la propagación de la solicitud enviando una respuesta al cliente.
- Un middleware puede enviar la respuesta a la solicitud utilizando uno de los métodos de respuesta en el objeto de respuesta o pasar la solicitud al siguiente middleware llamando a *next()*.
- Si un middleware envía la respuesta del servidor, convencionalmente, la solicitud no se propaga más en la cadena de middleware. Cualquier llamada a *next()* probablemente resultará en un error.
- Una función de middleware también puede tomar cuatro argumentos. En este caso, es un middleware de manejo de errores. Los parámetros de la función en su orden son: el objeto de error (err), el objeto de solicitud (req), el objeto de respuesta (res) y el siguiente middleware en la cadena (next).

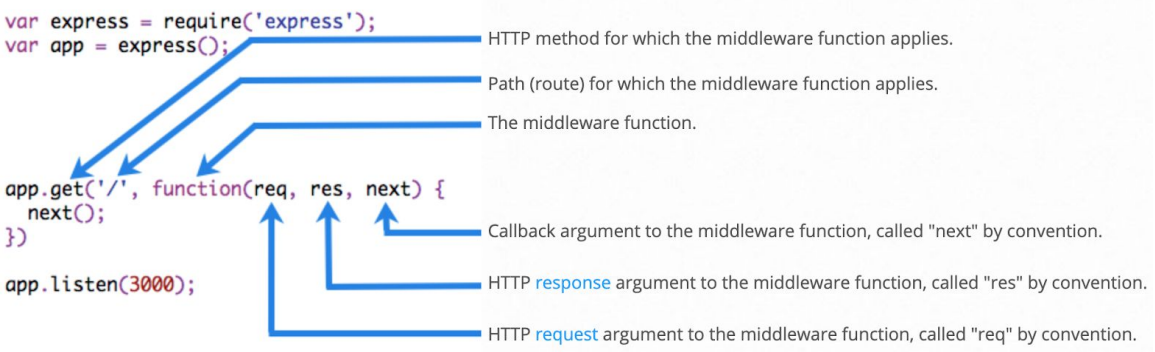

Fig. 15: Elementos de una llamada a una función de middleware.

4.4 React

React es la librería que se utilizó para desarrollar el frontend de ONE. Fue creada por Facebook para abordar algunos de los desafíos asociados con los sitios web basados en datos a gran escala. Cuando React se lanzó en 2013, el proyecto se vio inicialmente con cierto escepticismo porque las convenciones de React son bastante únicas.

Para poder trabajar con React en el navegador, es necesario incluir dos librerías: React y ReactDOM. React es la librería para crear vistas. ReactDOM es la librería que se usa para representar la interfaz de usuario en el navegador.

4.4.1 El DOM virtual

HTML es simplemente un conjunto de instrucciones que sigue un navegador al construir el modelo de objeto de documento o DOM. Los elementos que forman un documento HTML se convierten en elementos DOM cuando el navegador carga HTML y presenta la interfaz del usuario.

En HTML, los elementos se relacionan entre sí en una jerarquía que se asemeja a un árbol genealógico.

Tradicionalmente, los sitios web han consistido en páginas HTML independientes. Cuando el usuario navegaba por estas páginas, el navegador solicitaría y cargaría diferentes documentos HTML. La invención de AJAX nos trajo la aplicación de una sola página, o SPA. Dado que los navegadores pueden solicitar y cargar pequeños bits de datos con AJAX, las aplicaciones web completas ahora pueden componerse de una sola página y confiar en JavaScript para actualizar la interfaz de usuario.

En un SPA, el navegador carga inicialmente un documento HTML. A medida que los usuarios navegan por el sitio, permanecen en la misma página. JavaScript destruye y crea una nueva interfaz de usuario a medida que el usuario interactúa con la aplicación. Puede parecer que se está saltando de una página a otra, pero en realidad aún se está en la misma página HTML y JavaScript está haciendo el trabajo más pesado.

El DOM API es una colección de objetos que JavaScript puede usar para interactuar con el navegador para modificar el DOM. Actualizar o cambiar elementos DOM representados en JavaScript es relativamente fácil. Sin embargo, el proceso de inserción de nuevos elementos es extremadamente lento. Esto significa que si los desarrolladores web son meticulosos sobre cómo realizan cambios en la interfaz de usuario, pueden mejorar el rendimiento de sus aplicaciones.

El manejo eficiente de los cambios de DOM con JavaScript puede ser muy complicada y requerir mucho tiempo. Desde una perspectiva de codificación, es más fácil eliminar a todos los elementos secundarios de un elemento en particular y reconstruirlos que dejarlos en su lugar e intentar actualizarlos de manera eficiente. El problema es que es posible que no se tenga el tiempo o el conocimiento avanzado de JavaScript para trabajar de manera eficiente con la API DOM cada vez que se construye una nueva aplicación.

React es una librería diseñada para actualizar el DOM del navegador. Ya no es necesario preocuparse por las complejidades asociadas con la construcción de SPA de alto rendimiento porque React puede hacerlo por sí solo. Con React, no se interactúa directamente con la API DOM, en su lugar, se interactúa con un DOM virtual, o un conjunto de instrucciones que React utilizará para construir la interfaz de usuario e interactuar con el navegador.

El DOM virtual está formado por elementos React, que conceptualmente parecen similares a los elementos HTML, pero en realidad son objetos de JavaScript. Es mucho más rápido trabajar directamente con objetos de JavaScript que trabajar con la API DOM.

4.4.2 Elementos React

El DOM del navegador está formado por elementos DOM. Del mismo modo, el React DOM está formado por elementos React. Los elementos DOM y los elementos React pueden parecer iguales, pero en realidad son bastante diferentes. Un elemento React es una descripción de cómo debería verse el elemento DOM real. En otras palabras, los elementos React son las instrucciones sobre cómo se debe crear el DOM del navegador.

Podemos crear un elemento React para representar un *h1* usando *React.createElement*:

```
React.createElement("h1",
    {id: "recipe-0", 'data-type': "title"},
    "Baked Salmon"

)

<h1 data-reactroot id="recipe-0" data-type="title">Baked Salmon</h1>
```

Fig. 16: Ejemplo de creación de elemento React.

El primer argumento define el tipo de elemento que se desea crear. En este caso, un elemento de encabezado-uno. El segundo argumento representa las propiedades del elemento, las cuales son agregadas como atributos del tag. El tercer argumento representa los elementos secundarios del elemento, los nodos que se insertan entre el tag de apertura y de cierre.

4.4.3 ReactDOM

ReactDOM contiene las herramientas necesarias para representar los elementos React en el navegador. ReactDOM es donde se encuentra el método *render* y los métodos *renderToString* y *renderToStaticMarkup* que se utilizan en el servidor. Todas las herramientas necesarias para generar HTML desde el DOM virtual se encuentran en esta librería.

Es posible renderizar un elemento React, incluyendo sus hijos, en el DOM con *ReactDOM.render*. El elemento que se desea representar se pasa como el primer argumento y el segundo argumento es el nodo de destino, donde se debe renderizar el elemento:

```
var dish = React.createElement("h1", null, "Baked Salmon")

ReactDOM.render(dish, document.getElementById('react-container'))
```

Fig. 17: Ejemplo de renderizar un elemento React.

La principal ventaja de usar React es su capacidad para separar los datos de los elementos de la interfaz de usuario. Dado que React es solo JavaScript, podemos agregar lógica de JavaScript para ayudarnos a construir el árbol de componentes React.

```
var items = [
    "1 lb Salmon",
    "1 cup Pine Nuts",
    "2 cups Butter Lettuce",
    "1 Yellow Squash",
    "1/2 cup Olive Oil",
    "3 cloves of Garlic"
]

React.createElement("ul", {className: "ingredients"},
    items.map((ingredient, i) =>
        React.createElement("li", { key: i }, ingredient)
)
```

Fig. 18: Ejemplo de crear un elemento React con datos.

4.4.4 Componentes React

Cada interfaz de usuario se compone de partes. En React, describimos cada una de estas partes como un componente. Los componentes nos permiten reutilizar la misma estructura DOM para diferentes conjuntos de datos.

Los componentes son objetos que se pueden usar para encapsular código al igual que las clases. Nos permiten usar datos para construir una IU reutilizable. Los datos se pueden pasar a los componentes React como propiedades. En la función *render*, podemos usar la palabra clave *this* para referirse a la instancia del componente, y se puede acceder a las propiedades en esa instancia con *this.props.*

```
class IngredientsList extends React.Component {

  renderListItem(ingredient, i) {
    return React.createElement("li", { key: i }, ingredient)
  }

  render() {
    return React.createElement("ul", {className: "ingredients"},
        this.props.items.map(this.renderListItem)
    )
  }

}
```

Fig. 19: Ejemplo de componente React.

4.4.4.1 Componentes funcionales sin estado

Los componentes funcionales sin estado son funciones, no objetos. Debido a que son funciones simples y puras, se recomienda utilizarlos tanto como sea posible. Puede llegar un punto en el que el componente funcional sin estado no sea lo suficientemente robusto y se deba recurrir al uso de clases.

Los componentes funcionales sin estado son funciones que toman propiedades y devuelven un elemento DOM. Los componentes funcionales sin estado son una buena manera de practicar las reglas de la programación funcional. Debe esforzarse por hacer de cada componente funcional sin estado una función pura. Deben utilizar propiedades y devolver un elemento DOM sin causar efectos secundarios, fomentando la simplicidad.

```
const IngredientsList = props =>
    React.createElement("ul", {className: "ingredients"},
        props.items.map((ingredient, i) =>
            React.createElement("li", { key: i }, ingredient)
        )
    )
```

Fig. 20: Ejemplo de componente React sin estado.

4.4.5 React con JSX

El DOM virtual es un conjunto de instrucciones que React sigue al crear y actualizar una interfaz de usuario. Estas instrucciones se componen de objetos JavaScript llamados *elementos React*.

Una alternativa a escribir *React.createElement* es utilizar JSX, una extensión de JavaScript que nos permite definir los elementos de React usando una sintaxis similar a HTML.

El equipo React de Facebook lanzó JSX cuando lanzó React para proporcionar una sintaxis concisa para crear árboles DOM complejos con atributos. También esperaban hacer que React sea más legible, como HTML y XML.

En JSX, el tipo de un elemento se especifica con una etiqueta. Los atributos de la etiqueta representan las propiedades. Los elementos secundarios del elemento se pueden agregar entre las etiquetas de apertura y cierre. También se pueden agregar otros elementos JSX como hijos.

JSX funciona con componentes también. Simplemente se define el componente usando el nombre de la clase.

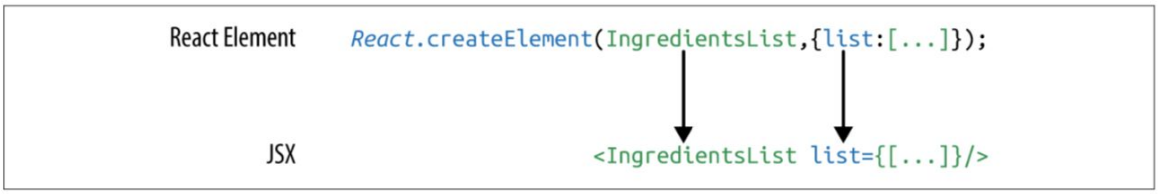

Fig. 21: Creación de un elemento React con JSX.

En el ejemplo de la Fig. 21, cuando se pasa la lista de ingredientes a este componente, ser debe rodearlo con llaves. Esto se denomina expresión de JavaScript, y se debe usar al pasar valores de JavaScript a los componentes como propiedades. Las propiedades de los componentes tendrán dos tipos: una cadena o una expresión de JavaScript. Las expresiones de JavaScript pueden incluir arreglos, objetos e incluso funciones.

4.4.5.1 Babel

La mayoría de los lenguajes de software necesitan compilar el código fuente. JavaScript es un lenguaje interpretado: el navegador interpreta el código como texto, por lo que no es necesario compilar. Sin embargo, ningún navegador admite la sintaxis de JSX. Para esto, es necesario convertir el código fuente de fantasía en algo que el navegador pueda interpretar. Este proceso se llama transpiling y Babel está diseñado para hacerlo.

La primera versión del proyecto se llamó 6to5 y se lanzó en septiembre de 2014. 6to5 era una herramienta que se podía usar para convertir la sintaxis ES6 a la sintaxis ES5, que es más ampliamente compatible con los navegadores web. A medida que el proyecto creció, pretendía ser una plataforma para soportar todos los cambios más recientes en ECMAScript. También creció para soportar la transmisión de JSX a React puro. El proyecto pasó a llamarse Babel en febrero de 2015.

Babel se utiliza en la producción en Facebook, Netflix, PayPal, Airbnb y más. Anteriormente, Facebook había creado un transformador JSX que era su estándar, pero pronto se retiró a favor de Babel.

Hay muchas maneras de trabajar con Babel. La forma más fácil de comenzar es incluir un enlace al transpiler de *babel-core* directamente en el HTML, que transpilará[7] cualquier código en bloques de script que tengan un tipo de "texto/babel". Babel transpilará el código fuente en el cliente antes de ejecutarlo. Si bien esta puede no ser la mejor solución para la producción, es una excelente manera de comenzar a utilizar JSX.

4.4.5.2 Webpack

Cuando se comienza a trabajar en producción con React, hay varias cuestiones para considerar: ¿Cómo lidiar con la transformación de JSX? ¿Cómo manejar las dependencias? ¿Cómo optimizar las imágenes y CSS?

Han surgido muchas herramientas diferentes para responder estas preguntas, incluyendo Browserify, Gulp y Grunt. Debido a sus características y su amplia adopción por parte de grandes empresas, *webpack* también se ha convertido en una de las herramientas líderes para agrupar módulos de CommonJS.

Webpack es un agrupador de módulos, toma todos archivos diferentes (JavaScript, MENOS, CSS, JSX, ES6, etc.) y los convierte en un solo archivo. Los dos beneficios principales de la agrupación modular son la modularidad y el rendimiento de la red.

La modularidad permite desglosar el código fuente en partes o módulos con los que sea más fácil trabajar, especialmente en un entorno de equipo.

El rendimiento de la red se obtiene solo con la necesidad de cargar una dependencia en el navegador, el paquete. Cada etiqueta de script realiza una solicitud HTTP y hay una penalización de latencia para cada solicitud HTTP. Agrupar todas las dependencias en un solo archivo le permite cargar todo con una solicitud HTTP, evitando así latencia adicional.

[7] Transpilar: proceso de transformar código de un lenguaje de programación a otro, similar a compilar, pero la diferencia está en que el código que se transforma, tienen el mismo nivel de abstracción.

Aparte de la transpilación, *Webpack* también puede manejar:

- **División de código:** Divide su código en diferentes trozos que pueden cargarse cuando se los necesite. A veces estos se llaman *rollups* o *capas*. El objetivo es dividir el código según sea necesario para diferentes páginas o dispositivos.
- **Minificación:** Elimina espacios en blanco, saltos de línea, nombres de variables largos y códigos innecesarios para reducir el tamaño del archivo.
- **Feature flagging:** Envía código a uno o más entornos, pero no a todos, al probar las distintos features.
- **Hot Module Replacement:** Vigila los cambios en el código fuente. Cambia solo los módulos actualizados inmediatamente.

El uso de una herramienta como webpack para construir estáticamente el JavaScript del cliente hace posible que los equipos trabajen juntos en aplicaciones web a gran escala. Algunos de los beneficios que se obtienen al incorporar el paquete de módulos de Webpack son:

- **Modularidad:** El uso del patrón del módulo CommonJS para exportar módulos que luego serán importados o requeridos por otra parte de la aplicación hace que el código fuente sea más accesible. Permite que los equipos de desarrollo trabajen juntos fácilmente, permitiéndoles crear y trabajar con archivos separados que se combinarán estáticamente en un solo archivo antes de enviarlos a producción.
- **Composición:** Con los módulos es posible construir componentes React pequeños, simples y reutilizables que podemos componer eficientemente en aplicaciones. Los componentes más pequeños son más fáciles de combinar, probar y reutilizar así como también son más fáciles de reemplazar.
- **Velocidad:** Empaquetar todos los módulos y dependencias de la aplicación en un solo paquete de clientes reducirá el tiempo de carga de la aplicación porque hay una latencia asociada con cada solicitud HTTP. Empaquetar todo en un solo archivo significa que el cliente solo tendrá que realizar una

única solicitud. Minificar el código en el paquete también mejorará el tiempo de carga.

- **Consistencia:** Babel admite una amplia gama de sintaxis de ESNext, lo que significa que no tenemos que preocuparnos de si el navegador es compatible con nuestro código. Permite a los desarrolladores utilizar de forma sistemática la sintaxis de JavaScript.

4.4.5.2.1 Webpack Loaders

Un *loader* es una función que maneja las transformaciones a través de las cuales se quiere poner el código durante el proceso de construcción. Si una aplicación utiliza JSX, CoffeeScript y algún otro idioma que el navegador no puede leer de forma nativa, se deben especificar los *loaders* necesarios en el archivo *webpack.config.js* para realizar el trabajo de conversión del código en una sintaxis que pueda ser leída de forma nativa por el navegador.

Webpack tiene una gran cantidad de *loaders* que caen en unas pocas categorías. El caso de uso más común para los *loaders* es la transpilación de un dialecto a otro.

Otra categoría popular de *loaders* es para el estilo. El *css-loader* busca archivos con la extensión .scss y los compila en CSS y se puede usar para incluir módulos CSS en su paquete. Todo el CSS se incluye como JavaScript y se agrega automáticamente cuando se incluye el archivo JavaScript incluido.

4.4.6 React State

Anteriormente hemos mostrado el uso de propiedades para manejar datos en los componentes React. Dichas propiedades son inmutables, una vez renderizadas no cambian. Para que nuestra interfaz de usuario cambie, necesitamos algún otro mecanismo que pueda volver a renderizar el árbol de componentes con nuevas propiedades. El estado o state en React es una opción integrada para administrar datos que cambiarán dentro de un componente. Cuando cambia el estado de la aplicación, la IU se vuelve a enviar para reflejar esos cambios.

Los usuarios interactúan con las aplicaciones, navegan, buscan, filtran, seleccionan, agregan, actualizan y eliminan. Cuando un usuario interactúa con una aplicación, el estado de esa aplicación cambia, y esos cambios se reflejan de nuevo al usuario en la interfaz de usuario. En React, la interfaz de usuario es un reflejo del estado de la aplicación.

El estado se puede expresar en componentes React con un solo objeto JavaScript. Cuando el estado de un componente cambia, el componente presenta una nueva interfaz de usuario que refleja esos cambios. ¿Qué puede ser más funcional que eso? Dados algunos datos, un componente de React representará esos datos como la interfaz de usuario. Dado un cambio en esos datos, React actualizará la interfaz de usuario de la manera más eficiente posible para reflejar ese cambio.

```
class StarRating extends Component {

    constructor(props) {
        super(props)
        this.state = {
            starsSelected: 0
        }
        this.change = this.change.bind(this)
    }

    change(starsSelected) {
        this.setState({starsSelected})
    }

    render() {
        const {totalStars} = this.props
        const {starsSelected} = this.state
        return (
            <div className="star-rating">
                {[...Array(totalStars)].map((n, i) =>
                    <Star key={i}
                          selected={i<starsSelected}
                          onClick={() => this.change(i+1)}
                    />
                )}
                <p>{starsSelected} of {totalStars} stars</p>
            </div>
        )
    }

}
```

Fig. 22: Ejemplo de componente que utiliza el state de React.

Todos los componentes de React pueden tener su propio estado, pero la pregunta es ¿Deberían tenerlo? La gracia de usar React no proviene de tener las variables de estado esparcidas en toda la aplicación. La gracia de usar React proviene de la creación de aplicaciones escalables que sean fáciles de entender. Lo más importante que se puede hacer para que una aplicación sea fácil de entender es limitar la cantidad de componentes que utilizan el estado tanto como sea posible.

En muchas aplicaciones React, es posible agrupar todos los datos de estado en el componente raíz. Los datos de estado se pueden pasar por el árbol de componentes a través de las propiedades, y los datos se pueden pasar de vuelta por el árbol a la raíz a través del enlace de función de dos vías. El resultado es que todo el estado de la aplicación completa existe en un solo lugar. Esto a menudo se conoce como tener una “fuente única de verdad”.

4.4.7 Ciclo de vida de los componentes

El ciclo de vida de los componentes consiste en métodos que se invocan en serie cuando se monta o actualiza un componente. Estos métodos se invocan antes o después de que el componente renderice la interfaz de usuario. De hecho, el método de renderización en sí mismo es parte del ciclo de vida del componente. Hay dos ciclos de vida principales: el ciclo de vida de montaje y el ciclo de vida de actualización.

4.4.7.1 Ciclo de vida de montaje

El ciclo de vida del montaje consiste en métodos que se invocan cuando un componente se monta o se desmonta. En otras palabras, estos métodos permiten configurar inicialmente el estado, hacer llamadas a la API, iniciar y detener temporizadores, manipular el DOM renderizado, inicializar bibliotecas de terceros y más. Estos métodos permiten incorporar JavaScript para ayudar en la inicialización y destrucción de un componente.

El ciclo de vida de montaje es ligeramente diferente dependiendo de si utiliza *React.createClass* o la sintaxis de clase ES6 o superior para crear componentes. Cuando utiliza *createClass*, primero se invoca *getDefaultProps* para obtener las

propiedades del componente. A continuación, se invoca *getInitialState* para inicializar el estado.

Las clases de ES no tienen estos métodos. En su lugar, las propiedades por defecto se obtienen y envían al constructor como un argumento. El constructor es donde se inicializa el estado. Tanto los constructores de clase ES como *getInitialState* tienen acceso a las propiedades y, si es necesario, pueden usarlos para ayudar a definir el estado inicial.

Técnicamente, el constructor no es un método de ciclo de vida. Se lo incluye ya que se utiliza para la inicialización de componentes (aquí es donde se inicializa el estado). Además, el constructor es siempre la primera función que se invoca cuando se monta un componente.

ES6 class	React.createClass()
	`getDefaultProps()`
`constructor(props)`	`getInitialState()`
`componentWillMount()`	`componentWillMount()`
`render()`	`render()`
`componentDidMount()`	`componentDidMount()`
`componentWillUnmount()`	`componentWillUnmount()`

Fig. 23: El ciclo de vida de montaje de componentes.

Una vez que se obtienen las propiedades y se inicializa el estado, se invoca el método *componentWillMount*. Este método se invoca antes de que se genere el DOM y se puede usar para inicializar bibliotecas de terceros, iniciar animaciones, solicitar datos o realizar cualquier configuración adicional que pueda ser necesaria antes de renderizar un componente. Es posible invocar *setState* desde este método para cambiar el estado del componente justo antes de que el componente se renderice inicialmente.

El método *componentDidMount* se invoca justo después de que el componente se haya procesado y es otro buen lugar para realizar solicitudes de API. Al ser invocado después de que el componente se haya procesado, cualquier llamada a *setState* de este método iniciará el ciclo de vida de la actualización y volverá a enviar el componente.

El método *componentWillUnmount* se invoca justo antes de que se desmonte el componente. Los componentes se desmontan cuando sus padres los eliminan o se desmontan con la función *unmountComponentAtNode* que se encuentra en react-dom. Este método se utiliza para desmontar el componente raíz. Cuando se desmonta un componente raíz, sus hijos se desmontan primero.

4.4.7.2 Ciclo de vida de actualización

El ciclo de vida de actualización es una serie de métodos que se invocan cuando cambia el estado de un componente o cuando se reciben nuevas propiedades del padre. Este ciclo de vida se puede usar para incorporar JavaScript antes de las actualizaciones del componente o para interactuar con el DOM después de la actualización. Además, se puede utilizar para mejorar el rendimiento de una aplicación porque le permite cancelar actualizaciones innecesarias.

El ciclo de vida de la actualización se inicia cada vez que se llama a *setState*. Llamar a *setState* dentro del ciclo de vida de actualización causará un bucle recursivo infinito que resultará en un error de desbordamiento de pila. Por lo tanto, *setState* solo se puede llamar en *componentWillReceiveProps*, que permite al componente actualizar el estado cuando sus propiedades están actualizadas.

Los métodos de actualización del ciclo de vida incluyen:

- ***componentWillReceiveProps (nextProps):*** Solo se invoca si se han pasado nuevas propiedades al componente. Este es el único método donde se puede llamar a *setState*.
- ***shouldComponentUpdate (nextProps, nextState):*** El controlador de acceso del ciclo de vida de la actualización: un predicado que puede cancelar la actualización. Este método se puede utilizar para mejorar el rendimiento permitiendo solo las actualizaciones necesarias.
- ***componentWillUpdate (nextProps, nextState):*** Se invoca justo antes de que se actualice el componente. Similar a *componentWillMount*, solo se invoca antes de que ocurra cada actualización.
- ***componentDidUpdate (prevProps, prevState):*** Se invoca justo después de que se realice la actualización, después de la llamada a representar.

Similar a *componentDidMount*, pero se invoca después de cada actualización.

4.4.8 Flux

Flux es un patrón de diseño desarrollado por Facebook que fue diseñado para mantener los datos fluyendo en una dirección. Antes de que se introdujera Flux, la arquitectura de desarrollo web estaba dominada por las variaciones del patrón de diseño MVC. Flux es una alternativa a MVC, un patrón de diseño completamente diferente que complementa el enfoque funcional.

¿Qué tiene que ver React o Flux con JavaScript funcional? Para empezar, un componente funcional sin estado es una función pura. Toma instrucciones como propiedades y devuelve elementos de la interfaz de usuario. Una clase React usa el estado o las propiedades como entrada y también producirá elementos de UI. Los componentes de React están compuestos en un solo componente. Los datos inmutables proporcionan al componente entrada y salida a medida que se devuelven los elementos de la interfaz de usuario.

Flux proporciona una forma de diseñar aplicaciones web que complementan el funcionamiento de React. Específicamente, Flux otorga una manera de proporcionar los datos que React utilizará para crear la interfaz de usuario.

En Flux, los datos del estado de la aplicación se administran fuera de los componentes React en las tiendas o *stores*. Las tiendas retienen y cambian los datos, y son lo único que puede actualizar una vista en Flux. Si un usuario interactuara con una página web (por ejemplo, haga clic en un botón o envíe un formulario), se creará una acción para representar la solicitud del usuario. Una acción proporciona las instrucciones y los datos necesarios para realizar un cambio. Las acciones se envían utilizando un componente de control central llamado el despachador o *dispatcher*. El despachador está diseñado para poner en cola las distintas acciones y enviarlas a la tienda adecuada. Una vez que una tienda recibe una acción, la usará como instrucciones para modificar el estado y actualizar la vista. Los datos fluyen en una dirección como lo muestra la Fig. 24.

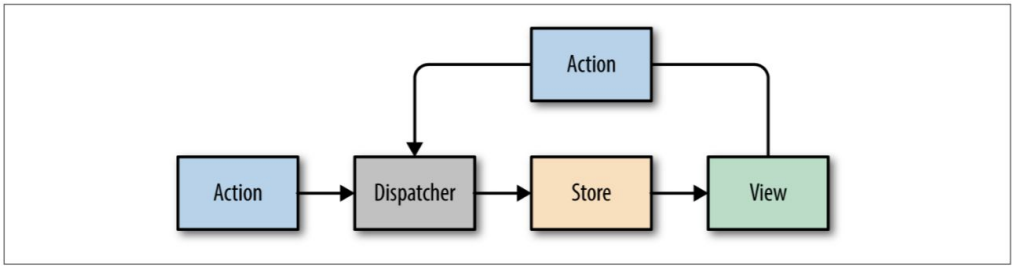

Fig. 24: Flujo de datos en Flux.

Las acciones y los datos de estado son inmutables en Flux. Las acciones se pueden enviar desde una vista o pueden provenir de otras fuentes, como por ejemplo de un servidor web.

Cada cambio requiere una acción. Cada acción proporciona las instrucciones para hacer el cambio. Las acciones también sirven como comprobantes de qué ha cambiado, qué datos se utilizaron para realizar el cambio y dónde se originó la acción. Este patrón no causa efectos secundarios ya que lo único que puede hacer un cambio es una tienda. Las tiendas actualizan los datos, las vistas representan esas actualizaciones en la interfaz de usuario y las acciones nos dicen cómo y por qué se han producido los cambios.

La restricción del flujo de datos de este patrón de diseño hace que la aplicación sea mucho más fácil de arreglar y escalar.

A continuación se presenta cada parte del patrón de diseño:

- **Vistas:** Generalmente compuestas por componentes sin estado de React. Flux administrará el estado de la aplicación, por lo que, a menos que se necesite una función de ciclo de vida, no es necesario utilizar componentes de clase.
- **Acciones y Creadores de acciones:** Las acciones proporcionan las instrucciones y los datos que la tienda utilizará para modificar el estado. Los creadores de acciones son funciones que se pueden usar para abstraer los detalles esenciales necesarios para construir una acción. Las acciones en sí son objetos que, como mínimo, contienen un campo de tipo. El tipo de acción suele ser una cadena en mayúsculas que describe la

acción. Además, las acciones pueden empaquetar cualquier dato requerido por la tienda.

- **Despachador:** Solo hay un despachador, y representa la parte de control de tráfico aéreo de este patrón de diseño. El despachador toma la acción, la empaqueta con cierta información sobre dónde se generó la acción y la envía a la tienda apropiada o las tiendas que manejarán la acción.

  Aunque Flux no es un framework, Facebook hace una fuente de Dispatcher de código abierto que se puede usar. La forma en que se implementan los despachadores suele ser estándar, por lo que es mejor utilizar el despachador de Facebook en lugar de codificar uno nuevo.

  Cuando se crea una tienda, se registra con el despachador y comienza a escuchar las acciones. Cuando se envía una acción, se maneja en el orden en que se recibió y se envió a las tiendas apropiadas.

- **Tiendas:** Las tiendas son objetos que contienen la lógica de la aplicación y los datos de estado. Las tiendas son similares a los modelos en el patrón MVC, pero las tiendas no están restringidas a administrar datos en un solo objeto. Es posible crear aplicaciones Flux que consisten en una sola tienda que maneja muchos tipos de datos diferentes.

  Los datos del estado actual se pueden obtener de una tienda a través de propiedades. Todo lo que una tienda necesita para cambiar los datos de estado se proporciona en la acción. Una tienda manejará las acciones por tipo y cambiará sus datos en consecuencia. Una vez que se cambian los datos, la tienda emitirá un evento y notificará a las vistas que se hayan suscrito a la tienda que sus datos han cambiado.

Existen diferentes enfoques para la implementación de Flux. Algunas bibliotecas han sido de código abierto basadas en implementaciones específicas de este patrón de diseño. Aquí hay algunos enfoques para Flux que vale la pena mencionar:

- **Reflux:** Un enfoque simplificado del flujo de datos unidireccional que se centra en acciones, tiendas y vistas.

- **Flummox:** Una implementación de Flux que permite construir módulos de Flux a través de la extensión de las clases de JavaScript.
- **Fluxible:** Un framework Flux creado por Yahoo para trabajar con aplicaciones de flujo isomorfo.
- **Redux:** Una biblioteca similar a Flux que logra modularidad a través de funciones en lugar de objetos.
- **MobX:** Una biblioteca de administración del estado que utiliza observables para responder a los cambios en el estado.

Todas estas implementaciones tienen tiendas, acciones y un mecanismo de envío, y favorecen a los componentes React como la capa de vista. Todas son variaciones del patrón de diseño de Flux, que en su núcleo tiene que ver con el flujo de datos unidireccional. Redux se ha convertido rápidamente en uno de los frameworks de Flux más populares.

4.4.9 Redux

Redux se basa en Flux y fue diseñado para enfrentar el desafío de comprender cómo fluyen los cambios de datos a través de la aplicación. Redux fue desarrollado por Dan Abramov y Andrew Clark. Desde la creación de Redux, ambos han sido contratados por Facebook para trabajar en el equipo React.

Redux es sorprendentemente pequeño, solo 99 líneas de código. (Banks & Porcello, 2017).

Redux, al igual que Flux, tiene acciones, creadores de acciones, una tienda y objetos de acción que se utilizan para cambiar el estado. Pero Redux simplifica un poco los conceptos de Flux al eliminar el despachador y al representar el estado de la aplicación con un solo objeto inmutable. Redux también introduce reductores, que no son parte del patrón Flux. Los reductores son funciones puras que devuelven un nuevo estado en función del estado actual y una acción.

```
(state, action) => newState.
```

Fig. 25: Ejemplificación de reductor

4.4.9.1 Estado

En las aplicaciones React o Flux puras, se recomienda almacenar el estado en la menor cantidad de objetos posible.

En Redux, almacenar el estado en un único lugar es una regla.

Consideremos cómo se puede lograr esto con una aplicación que tiene muchos tipos diferentes de datos. Utilizaremos una aplicación de redes sociales que tiene un estado extendido a través de diferentes componentes (Fig. 26). La aplicación, los mensajes, las publicaciones y a su vez cada mensaje y publicación tienen su propio estado.

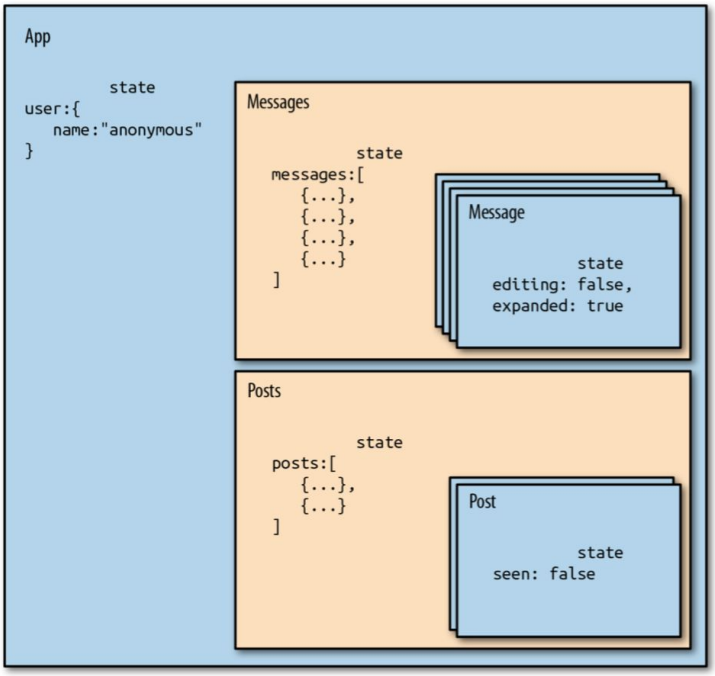

Fig. 26: Aplicación React donde cada componente tiene su propio estado.

Una aplicación estructurada como esta puede funcionar bien, pero a medida que crece, puede ser difícil determinar el estado general de la aplicación. También puede volverse incómodo entender de dónde vienen las actualizaciones, teniendo en cuenta que cada componente mutará su propio estado con llamadas internas a setState.

¿Qué mensajes se expanden? ¿Qué publicaciones se han leído? Para descubrir estos detalles, debemos sumergirnos en el árbol de componentes y rastrear el estado dentro de los componentes individuales.

Redux simplifica la forma en la que se ve el estado de la aplicación al exigir que se almacenen todos los datos de estado en un solo objeto. Todo lo que se necesita saber sobre la aplicación está en un solo lugar: una única fuente de verdad. Podríamos construir la misma aplicación con Redux moviendo todos los datos del estado a una sola ubicación (Fig. 27).

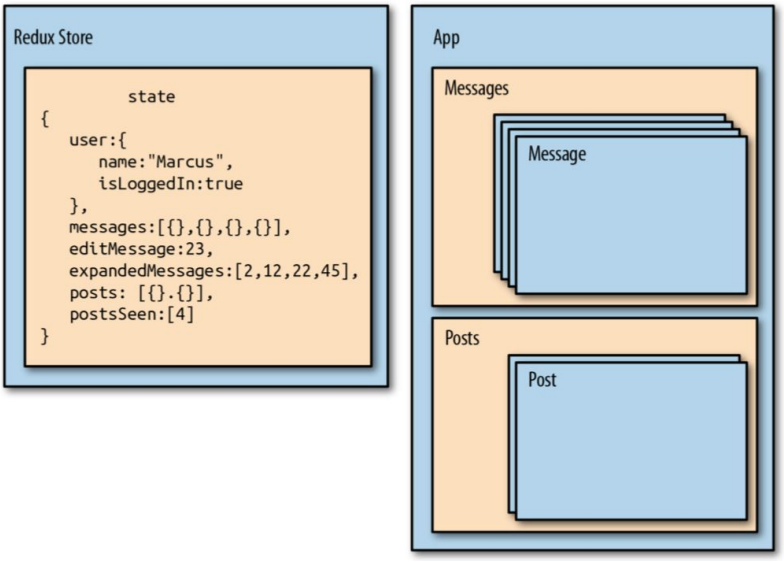

Fig. 27: Aplicación React donde el estado de la aplicación se encuentra centralizado.

En este último ejemplo, se puede observar que se está administrando el estado del usuario actual, los mensajes y las publicaciones desde el mismo objeto: la tienda Redux. Este objeto incluso almacena información sobre el mensaje que se está editando, qué mensajes se expanden y qué publicaciones se han visto. Esta información se captura en matrices que contienen ID que hacen referencia a registros específicos. Todos los mensajes y publicaciones se almacenan en caché en este objeto de estado, por lo que los datos están allí.

4.4.9.2 Acciones

Como se mencionó anteriormente, Redux tiene una regla muy fuerte e importante: el estado de la aplicación debe almacenarse en un solo objeto inmutable. Inmutable significa que este objeto de estado no cambia. Eventualmente se actualizará este objeto de estado reemplazándolo por completo. Para realizar esto, se utilizan las acciones: instrucciones sobre qué debe cambiar en el estado de la aplicación junto con los datos necesarios para realizar esos cambios.

Las acciones son la única forma de actualizar el estado de una aplicación Redux. Las acciones proporcionan instrucciones sobre lo que debería cambiar, pero también se las puede ver como marcas sobre el historial de lo que ha cambiado con el tiempo.

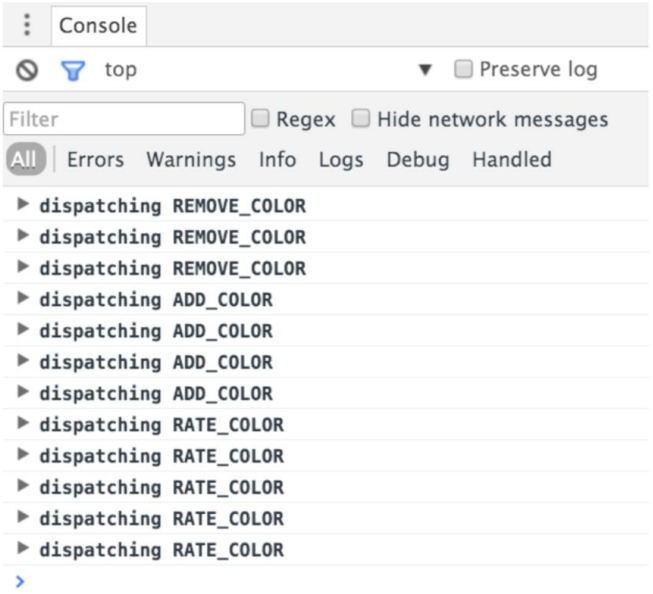

Fig. 28: Acciones que se registran en la consola a medida que se envían.

Generalmente, cuando se comienza a construir una aplicación orientada a objetos, se empieza por identificar los objetos, sus propiedades y cómo funcionan juntos. En este caso, está orientado al sustantivo. Cuando se construye una aplicación de Redux, hay que cambiar el pensamiento a orientado al verbo.

Las acciones son literales de JavaScript que proporcionan las instrucciones necesarias para hacer un cambio de estado. La mayoría de los cambios de estado también requieren algunos datos.

Nos referimos a estos datos como la carga útil (payload) de la acción. Esta información se puede pasar directamente con la acción en el mismo literal de JavaScript.

En síntesis, las acciones describen que algo pasó, pero no especifican cómo cambió el estado de la aplicación en respuesta. Esto es trabajo de los reducers.

```
{
   type: "RATE_COLOR",
   id: "a5685c39-6bdc-4727-9188-6c9a00bf7f95",
   rating: 4
}
```

Fig. 29: Ejemplo de acción.

4.4.9.3 Reducers

Ya que todo el árbol de estado se almacena en un solo objeto, es posible pensar que no es lo suficientemente modular. Redux logra modularidad a través de funciones que se utilizan para actualizar partes del árbol de estados. Estas funciones se llaman reducers.

Los reducers son funciones que toman el estado actual junto con una acción como argumentos y los utilizan para crear y devolver un nuevo estado. Los reducers están diseñados para actualizar partes específicas del árbol del estado, ya sea hojas o ramas.

Se llaman reducers ya que son el tipo de función que se podría llegar a pasar a *Array.prototype.reduce(reducer, initialValue)*. Es muy importante que los reducer se mantengan puros. Algunas cosas que nunca se debería hacer dentro de un reducer:

- Modificar sus argumentos.
- Realizar tareas con efectos secundarios como llamas a un API o transiciones de rutas.
- Llamar una función no pura, por ejemplo Date.now() o Math.random().

Los reducers deben ser funciones puras. Dados los mismos argumentos, deberían calcular y devolver el siguiente estado. Sin sorpresas. Sin efectos secundarios. Sin llamadas a APIs. Sin mutaciones. Solo cálculos.

```
function todoApp(state = initialState, action) {
  switch (action.type) {
    case SET_VISIBILITY_FILTER:
      return Object.assign({}, state, {
        visibilityFilter: action.filter
      })
    case ADD_TODO:
      return Object.assign({}, state, {
        todos: [
          ...state.todos,
          {
            text: action.text,
            completed: false
          }
        ]
      })
    case COMPLETE_TODO:
      return Object.assign({}, state, {
        todos: state.todos.map((todo, index) => {
          if(index === action.index) {
            return Object.assign({}, todo, {
              completed: true
            })
          }
          return todo
        })
      })
    default:
      return state
  }
}
```

Fig. 30: Ejemplo de reducers.

4.4.9.4 Stores

En Redux, el store o tienda es lo que contiene los datos de estado de la aplicación y maneja todas las actualizaciones de estado. Si bien el patrón de diseño de Flux permite que muchos stores se enfoquen en un conjunto específico de datos, Redux tiene solamente uno.

El store maneja las actualizaciones de estado pasando el estado actual y la acción a través de un reducer único.

```
import { createStore } from 'redux'
import { color } from './reducers'

const store = createStore(color)

console.log( store.getState() )           // {}
```

Fig. 31: Ejemplo de Store con reducers.

La única forma de cambiar el estado de la aplicación es mediante el envío de acciones a través de la tienda. La tienda tiene un método de envío que está listo para realizar acciones como un argumento. Cuando envía una acción a través de la tienda, la acción se envía a través de los reducers y el estado se actualiza.

```
console.log(
    "Length of colors array before ADD_COLOR",
    store.getState().colors.length
)

// Length of colors array before ADD_COLOR 3

store.dispatch({
  type: "ADD_COLOR",
  id: "2222e1p5-3abl-0p523-30e4-8001l8yf2222",
  title: "Party Pink",
  color: "#F142FF",
  timestamp: "Thu Mar 10 2016 01:11:12 GMT-0800 (PST)"
})

console.log(
    "Length of colors array after ADD_COLOR",
    store.getState().colors.length
)

// Length of colors array after ADD_COLOR 4
```

Fig. 32: Ejemplo actualización de estado mediante el Store.

El store permite suscribirse a las funciones que se invocan cada vez que él completa el envío de una acción.

```
store.subscribe(() =>
    console.log('color count:', store.getState().colors.length)
)

store.dispatch({
    type: "ADD_COLOR",
    id: "2222e1p5-3abl-0p523-30e4-8001l8yf2222",
    title: "Party Pink",
    color: "#F142FF",
    timestamp: "Thu Mar 10 2016 01:11:12 GMT-0800 (PST)"
})

store.dispatch({
    type: "ADD_COLOR",
    id: "3315e1p5-3abl-0p523-30e4-8001l8yf2412",
    title: "Big Blue",
    color: "#0000FF",
    timestamp: "Thu Mar 10 2016 01:11:12 GMT-0800 (PST)"
})


    // Console Output

    // color count: 1
    // color count: 2
    // color count: 2
    // color count: 1
```

Fig. 33: Ejemplo de suscripción a una función del Store.

4.4.9.5 Middleware

Redux utiliza un middleware, el cual sirve como unión entre dos capas diferentes o diferentes piezas de software.

El middleware de Redux consta de una serie de funciones que se ejecutan en fila en el proceso de despachar una acción, como se muestra en la Fig. 34.

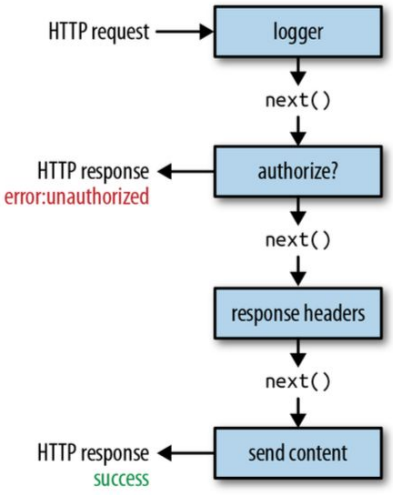

Fig. 34: Pipeline del middleware de HTTP request.

Estas funciones de orden superior permiten insertar funcionalidad antes o después de que se envíen las acciones y se actualice el estado. Cada función de middleware se ejecuta secuencialmente (Fig. 35).

Cada pieza de middleware es una función que tiene acceso a la acción, una función de despacho y una función que llamará a continuación. A continuación hace que se produzca la actualización. Antes de llamar a *next()*, es posible modificar la acción. Después de su llamado, el estado habrá cambiado.

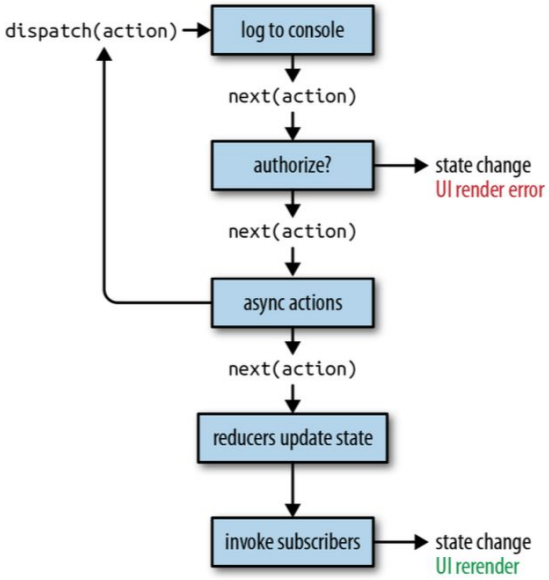

Fig. 35: Funciones de middleware.

4.5 REST

4.5.1 Breve historia de REST

Hoy en día, temas como la computación en la nube y las fuentes de servicio de dispositivos móviles y otras fuentes de datos alimentadas por tecnologías de vanguardia, escalables, sin estado y modernas, como los servicios web RESTful, dejan la impresión de que REST se ha inventado recientemente. Pero esto no es así ya que REST se definió a finales del siglo XX. Esto sucedió en 1999, cuando se envió una solicitud de comentarios al Grupo de trabajo de ingeniería de Internet (IETF: http://www.ietf.org/) a través del RFC 2616: "Protocolo de transferencia de hipertexto - HTTP/1.1". Uno de sus autores, Roy Fielding, definió

posteriormente un conjunto de principios basados en los estándares HTTP y URI. Esto dio a luz a REST como lo conocemos hoy en día.

Los principios clave en torno a los estándares de HTTP y URI que hacen que una aplicación se encuentre habilitada para el servicio REST son:

- Todo es un recurso.
- Cada recurso es identificable por un identificador único (URI).
- Utiliza los métodos HTTP estándar.
- Los recursos pueden tener múltiples representaciones.
- Comunicación sin estado.

4.5.1.1 Principio 1 - Todo es un recurso

Para comprender este principio, se debe concebir la idea de representar datos con un formato específico y no con un archivo físico. Cada dato disponible en Internet tiene un formato que podría describirse por un tipo de contenido. Algunos ejemplos de recursos y su tipo de contenido son:

- Imágenes JPEG -> image/jpeg
- Videos MPEG -> video/mpeg
- HTML -> text/html
- XML -> text/xml
- Datos binarios -> application/octet-stream

4.5.1.2 Principio 2 - Cada recurso es identificable por un identificador único

Dado que Internet contiene tantos recursos diferentes, todos deben ser accesibles a través de URI y deben identificarse de manera única. Además, los URI pueden estar en un formato legible para las personas, a pesar del hecho de que es más probable que sus consumidores sean programadores de software en lugar de los humanos comunes.

El URI mantiene los datos autodescriptivos y facilita su desarrollo. Además, el uso de URI legibles para el usuario ayuda a reducir al mínimo el riesgo de errores lógicos en los programas.

```
http://www.mydatastore.com/images/vacation/2014/summer
http://www.mydatastore.com/videos/vacation/2014/winter
http://www.mydatastore.com/data/documents/balance?format=xml
http://www.mydatastore.com/data/archives/2014
```

Fig. 36: Ejemplos de URIs.

4.5.1.3 Principio 3 - Utiliza los métodos HTTP estándar

El protocolo HTTP define nueve acciones, también conocidas como verbos:

- POST
- GET
- PUT
- DELETE
- HEAD
- OPTIONS
- TRACE
- CONNECT
- PATCH

Los primeros cuatro de ellos se sienten naturales en el contexto de los recursos, especialmente al definir acciones para la manipulación de datos de recursos. Si se compara con las bases de datos relacionales donde el lenguaje nativo para la manipulación de datos es CRUD podríamos decir que estos verbos son similares a las sentencias INSERT, SELECT, UPDATE y DELETE respectivamente.
Los verbos HTTP deberían utilizarse de la siguiente forma:

Verbo HTTP	**Acción**	**Status code**
GET	Solicitar un recurso	**"200 OK"** si existe el recurso, **"404 Not Found"** si no existe y **"500 Internal Server Error"** para otros errores
PUT	Crear o actualizar un recurso	**"201 CREATED"** sí se crea un nuevo recurso, **"200 OK"** si se

		actualiza y **"500 Internal Server Error"** para otros errores
POST	Actualizar un recurso existente	**"200 OK"** si el recurso se ha actualizado correctamente, **"404 Not Found"** si el recurso a actualizar no existe, y **"500 Internal Server Error"** para otros errores
DELETE	Eliminar un recurso	**"200 OK"** si el recurso se ha eliminado con éxito, **"404 Not Found"** si el recurso a eliminar no existe, y **"500 Internal Server Error"** para otros errores

4.5.1.4 Principio 4 - Los recursos pueden tener múltiples representaciones

Una característica clave de los recursos es que pueden estar representados en una forma diferente a la que se almacenan. Por lo tanto, puede ser solicitado o publicado en diferentes representaciones. Mientras se admita el formato especificado, el punto final habilitado para REST debe usarlo.

```
POST /data/documents/balance HTTP/1.1

Content-Type: text/xml
Host: www.mydatastore.com

<?xml version="1.0" encoding="utf-8"?>
<balance date="22082014">
    <Item>Sample item</Item>
    <price currency="EUR">100</price>
</balance>

HTTP/1.1 201 Created
Content-Type: text/xml
Location: /data/documents/balance
```

Fig. 37: Ejemplo de POST con XML.

```
POST /data/documents/balance HTTP/1.1

Content-Type: application/json
Host: www.mydatastore.com

{
    "balance": {
        "date": "22082014",
        "Item": "Sample item",
        "price": {
            "currency": "EUR",
            "text": "100"
        }
    }
}

HTTP/1.1 201 Created
Content-Type: application/json
Location: /data/documents/balance
```

Fig. 38: Ejemplo de POST con JSON.

4.5.1.5 Principio 5 - Comunicación sin estado

Las operaciones de manipulación de recursos a través de solicitudes HTTP siempre deben considerarse atómicas. Todas las modificaciones de un recurso deben llevarse a cabo dentro de una solicitud HTTP aislada. Después de la ejecución de la solicitud, el recurso se deja en un estado final, lo que implícitamente significa que las actualizaciones parciales de recursos no son compatibles. Siempre se debe enviar el estado completo del recurso.

Si nos basamos en el ejemplo de la *Fig. 38*, actualizar el campo de precio de un saldo dado significaría publicar un objeto JSON completo que contenga todos los datos del saldo, incluido el campo de precio actualizado. Publicar solo el precio actualizado implica que la aplicación es consciente de que el recurso tiene un campo de precio, es decir, que conoce su estado.

Otro requisito para una aplicación RESTful es no manejar estados, esto significa que cada solicitud es autodescriptiva y tiene suficiente contexto para que el

servidor procese ese mensaje. La falta de estado de una API RESTful aísla al cliente que llama contra los cambios en el lado del servidor.

La popularidad de REST se debe a que hasta el año 1999, para poder integrarse con las APIs la mayoría de los desarrolladores tenían que lidiar con SOAP, que traducido significa Protocolo Simple de Acceso a Objetos, pero que realmente no tenía nada simple. SOAP era conocido por ser complejo de construir, complejo de usar y casi imposible de depurar. Y la alternativa, CORBA, fue aún peor. El problema era que no había un estándar sobre cómo se deberían diseñar y usar las API. En aquel entonces, las API no estaban diseñadas para ser accesibles, solo estaban diseñadas para ser flexibles.

4.5.2 Los objetivos de REST

Presentados los principios de REST, se profundizará en lo que se puede lograr cuando se siguen.

4.5.2.1 Separación de la representación y el recurso

Un recurso es solo un conjunto de información y, como se define en el principio 4, puede tener múltiples representaciones. Sin embargo, el estado del recurso es atómico. Depende del cliente que llama especificar el encabezado de tipo de contenido de la solicitud http y luego depende de la aplicación del servidor manejar la representación en consecuencia y devolver el código de estado HTTP apropiado:

- HTTP 200 OK en caso de éxito.
- HTTP 400 Bad Request si se solicita un tipo de contenido no compatible, o para cualquier otra solicitud no válida.
- HTTP 500 Internal Server Error cuando ocurre algo inesperado durante el procesamiento de la solicitud.

Por ejemplo, si en el servidor, tenemos recursos almacenados en un archivo XML, podemos tener una API que permite a un consumidor solicitar el recurso en varios formatos, como application/json, application/zip, application/octet-stream, etc.

Es responsabilidad de la API cargar el recurso solicitado, transformarlo en el tipo solicitado (por ejemplo, JSON o XML) y usar zip para comprimirlo o vaciarlo directamente en la salida de respuesta HTTP. Es el encabezado HTTP de aceptación el que especifica la representación esperada de los datos de respuesta.

4.5.2.2. Visibilidad

REST está diseñado para ser visible y simple. La visibilidad del servicio significa que cada aspecto del mismo debe ser autodescriptivo y debe seguir el lenguaje HTTP natural según los principios 3, 4 y 5.

La visibilidad en el contexto del mundo exterior significa que las aplicaciones de monitoreo solo estarían interesadas en la comunicación HTTP entre el servicio REST y el cliente que llama. Dado que las solicitudes y respuestas son sin estado y atómicas, no se necesita nada más para hacer fluir el comportamiento de la aplicación y para comprender si algo salió mal.

Es necesario tener en cuenta que el almacenamiento en caché reduce la visibilidad de las aplicaciones REST y debería evitarse.

4.5.2.3 Confiabilidad

Antes de hablar de confiabilidad, es necesario definir qué métodos HTTP son seguros y cuáles son idempotentes en el contexto REST:

- Un método HTTP se considera seguro siempre que cuando se solicite, no modifique ni cause ningún efecto secundario en el estado del recurso.
- Se considera que un método HTTP es idempotente si su respuesta es siempre la misma, sin importar cuántas veces se solicite.

La siguiente tabla muestra qué método HTTP es seguro y cuál es idempotente:

Método HTTP	**Seguro**	**Idempotente**
GET	Si	Si
POST	No	No
PUT	No	Si
DELETE	No	Si

4.5.2.4 Escalabilidad y rendimiento

Hasta ahora, se ha enfatizado en la importancia de tener una implementación sin estado y un comportamiento sin estado para una aplicación web RESTful. La Web es un universo enorme, que contiene una gran cantidad de datos y usuarios ansiosos por obtener esos datos. Su evolución ha provocado el requisito de que las aplicaciones deben escalarse fácilmente a medida que aumenta su carga. Las aplicaciones de escalado que tienen un estado no son posibles, especialmente cuando se necesita un tiempo de inactividad de cero o casi cero.

Es por eso que no manejar un estado es crucial para cualquier aplicación que necesite escalar. En el mejor de los casos, escalar su aplicación requeriría que coloque otra pieza de hardware para un balanceador de carga. No habría necesidad de que los diferentes nodos se sincronicen entre sí, ya que no deberían preocuparse por el estado en absoluto. La escalabilidad se trata de servir a todos sus clientes en un tiempo aceptable. La idea principal es mantener la aplicación en ejecución y prevenir la denegación de servicio (DoS) causada por una gran cantidad de solicitudes entrantes.

La escalabilidad no debe confundirse con el rendimiento de una aplicación. El rendimiento se mide por el tiempo necesario para procesar una única solicitud, no por el número total de solicitudes que la aplicación puede manejar. La arquitectura asincrónica sin bloqueo y el diseño controlado por eventos de Node.js lo convierten en una opción lógica para implementar una aplicación bien escalable que funcione bien.

Los servicios REST se han convertido en proveedores de feeds de datos estándar de facto para servicios sociales, noticias y dispositivos móviles. Entregan una gran cantidad de datos a millones de usuarios. Por lo tanto, deben abordar los requisitos de alta disponibilidad, como la confiabilidad y la escalabilidad.

CAPÍTULO 5. ONE

La primera parte de este capítulo mostrará la arquitectura de ONE, identificando los componentes principales (desarrollados con las tecnologías descriptas en el Capítulo 4) y se detallarán los distintos sistemas con los cuales se integra para lograr publicar las campañas en las distintas plataformas.

La segunda parte del capítulo estará centrada en la interfaz de usuario desarrollada, haciendo hincapié en la experiencia de usuario, uno de los desafíos más complejos de ONE. Se expondrán los relevamientos realizados y los *wireframes* que surgieron de éstos, hasta la evolución final y actual de las pantallas que componen a ONE.

5.1 Arquitectura

El sistema ONE tiene tres componentes principales:

- Un frontend desarrollado en React el cual utiliza Material-UI como framework para la interfaz de usuario.
- Un backend desarrollado en Node.js utilizando LoopBack como framework.
- Un motor de reglas desarrollado en Java con el cual se realizan sugerencias y consejos en la plataforma, utilizando Drools y Spring Boot.

La Fig. 39 muestra un esquema sintetizado de la arquitectura de ONE.

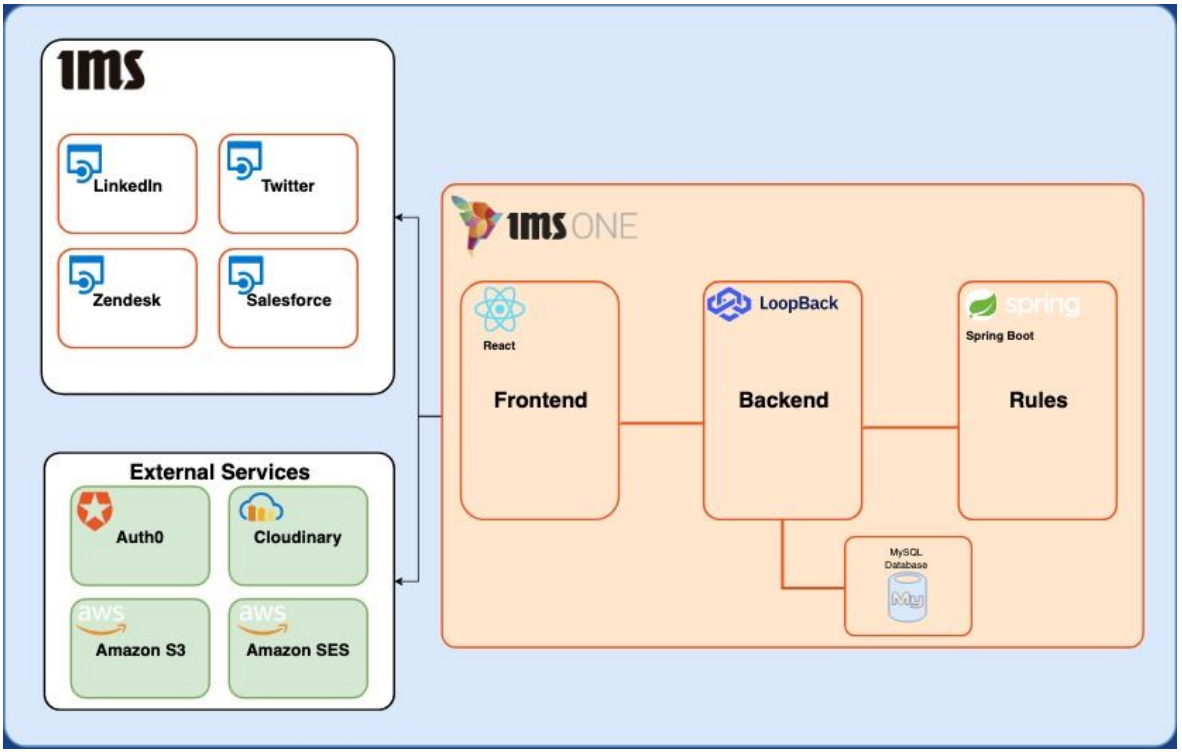

Fig. 39: Arquitectura de ONE.

Si bien estos componentes constituyen el corazón de ONE, necesitan de la integración con otros sistemas para su completo funcionamiento. Muchos de ellos son sistemas internos de IMS, como el CRM donde se registran las ventas, y otros totalmente externos, como Cloudinary donde se almacenan todos los recursos multimedia utilizados en ONE.

A continuación se detallarán los distintos sistemas con los cuales se integra, junto a la utilidad de los mismos en el proceso de carga de campañas:

- **Auth0:** Es una plataforma utilizada para la autenticación y administración de los usuarios que pueden utilizar ONE.
- **Cloudinary:** Es un gestor de archivos multimedia. Los distintos contenidos multimedia (audio, imagen, video) que componen a los creativos de las campañas son almacenados aquí.
- **Amazon S3:** Amazon Simple Storage Service es un servicio de almacenamiento de objetos. ONE lo utiliza para almacenar todos aquellos archivos que no son multimedia, como por ejemplo, las órdenes de compra.
- **Amazon SES:** Amazon Simple Email Service es un servicio de envío de correo electrónico. ONE lo utiliza para notificar a los usuarios de distintos

eventos, como por ejemplo, que una campaña fue publicada satisfactoriamente.

- **Zendesk:** Es el sistema de tickets interno de IMS. Cada vez que una orden de compra es enviada a validar o se solicita la publicación de una campaña, ONE crea un ticket en este sistema, lo que le permitirá a IMS revisar la información enviada y determinar si esta es correcta o no. La aprobación o rechazo de estas solicitudes son notificadas a ONE para que realice las acciones pertinentes ante estas situaciones. Por ejemplo, en el caso de aprobarse una solicitud de publicación, Zendesk notifica de esto a ONE y este último se encargará de publicar la campaña en la plataforma correspondiente.
- **Salesforce:** Es el CRM interno de IMS. Cada vez que una campaña es publicada, ONE registra en él una oportunidad o venta ganada, lo que permite a IMS llevar un control de sus ventas realizadas por este canal.
- **Twitter - LinkedIn:** Ambos son wrappers de las APIs de los sistemas de publicidad de Twitter y LinkedIn, desarrollados por IMS. ONE las utiliza tanto para obtener información, como en el caso de los valores posibles para las segmentaciones, así como también para crear, pausar, detener, reanudar o borrar una campaña y todo lo relacionado a ella. Estos wrappers fueron creados con dos objetivos principales:
    - Generar un almacén de datos históricos de las plataformas para poder asegurar la consistencia de la información con los datos generados en ONE, ya que los datos de las plataformas tienen alta volatilidad.
    - Mantener relacionada la información de las distintas plataformas que manejan sus clientes.

5.2 Experiencia de usuario

Experiencia de Usuario (UX, por sus siglas en inglés) refiere a un tipo de Diseño basado en la búsqueda de productos que resuelvan necesidades concretas de usuarios y que consigan la mejor experiencia y satisfacción de uso con el menor

esfuerzo, a través de un trabajo multidisciplinario y teniendo en cuenta la subjetividad de los distintos usuarios.

Para ONE la experiencia de usuario fue uno de los desafíos más complejos que tuvo que afrontar ya que tenía como objetivo lograr una experiencia similar o superior a la que ofrecían las plataformas de las redes sociales y al mismo tiempo debía motivar al personal de las agencias publicitarias a desear utilizarla.

Al comienzo del proyecto se realizó el relevamiento de información con los distintos stakeholders, desde account managers, comerciales y personal de las agencias publicitarias. De éste se desprendió el entendimiento del proceso de creación de campaña, de los distintos parámetros que la componen y cuáles de ellos hay en común entre todas las plataformas.

Luego del relevamiento inicial se realizó el primer prototipo en calidad baja o wireframe y fue validado por distintas agencias, oportunidad que sirvió para un segundo relevamiento mucho más detallado y rico en contenido.

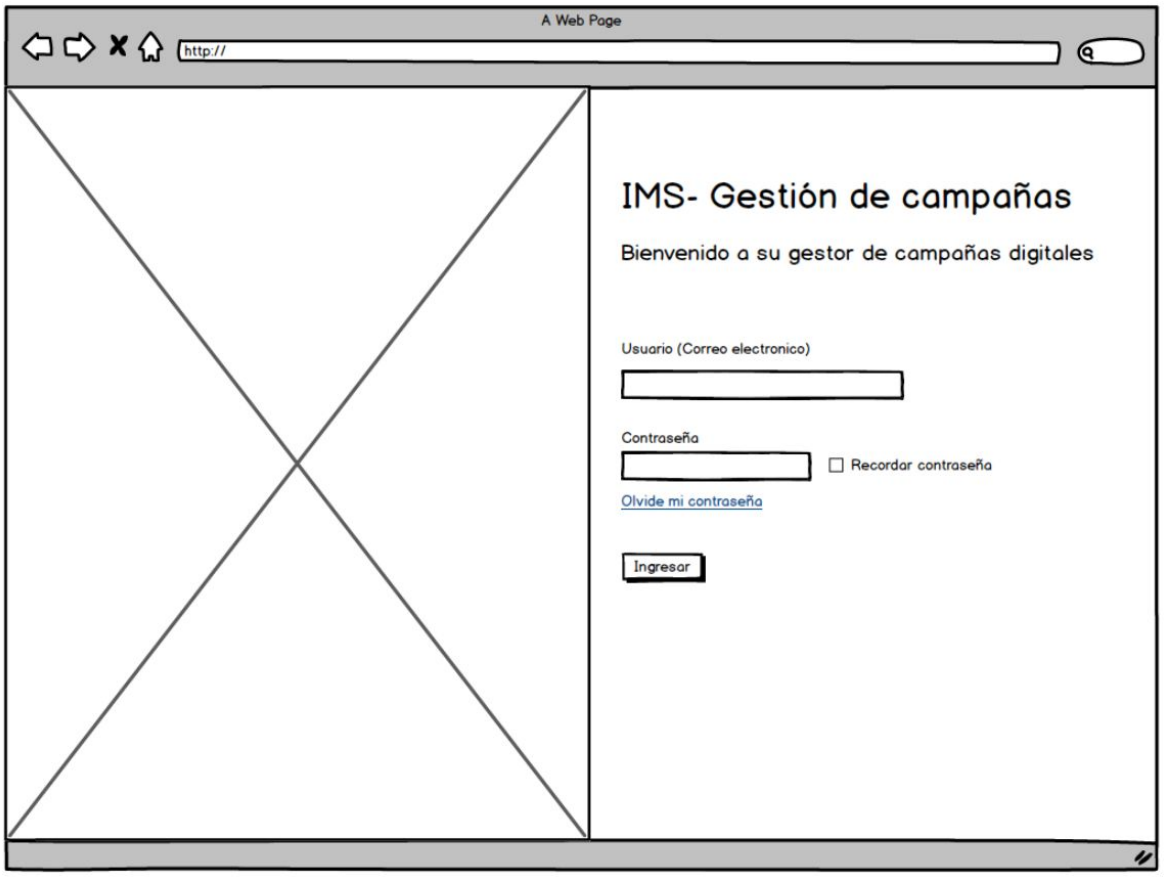

Fig. 40: Wireframe del login.

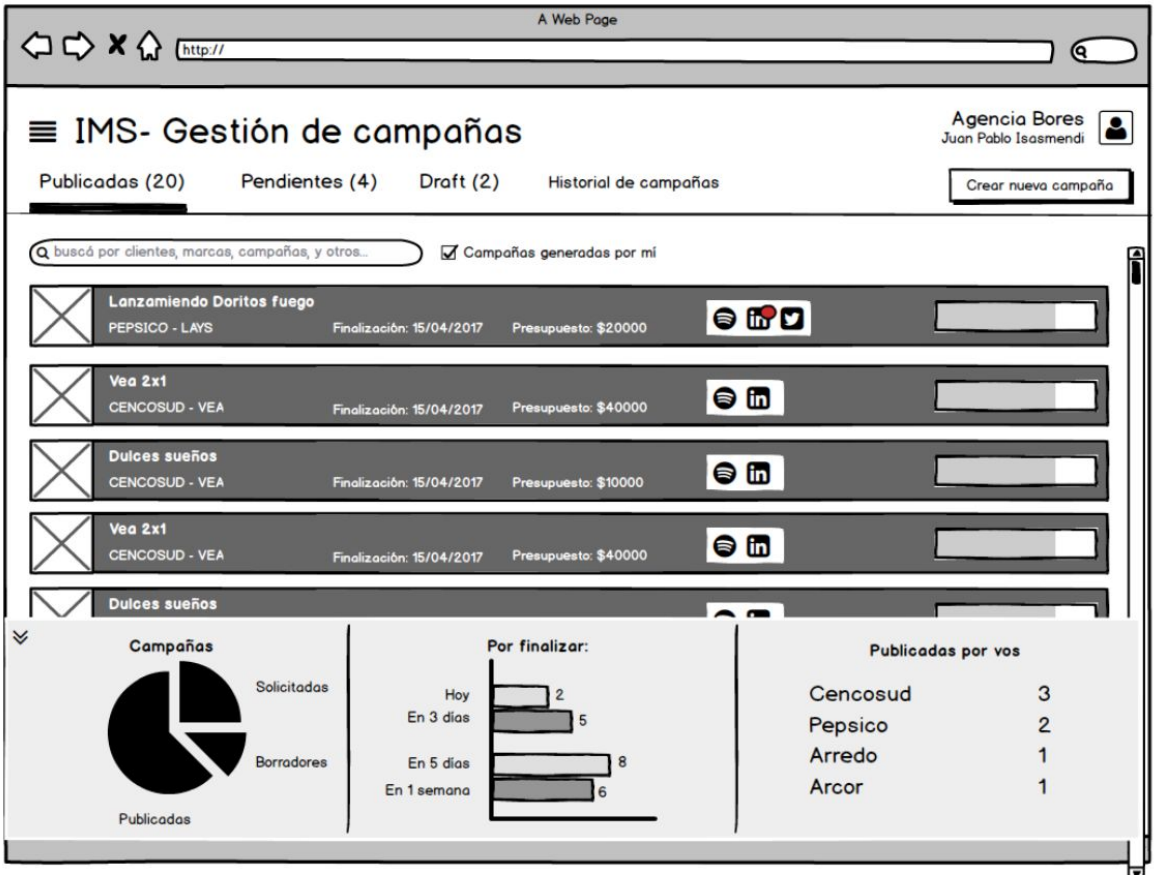

Fig. 41: Wireframe del listado de campañas.

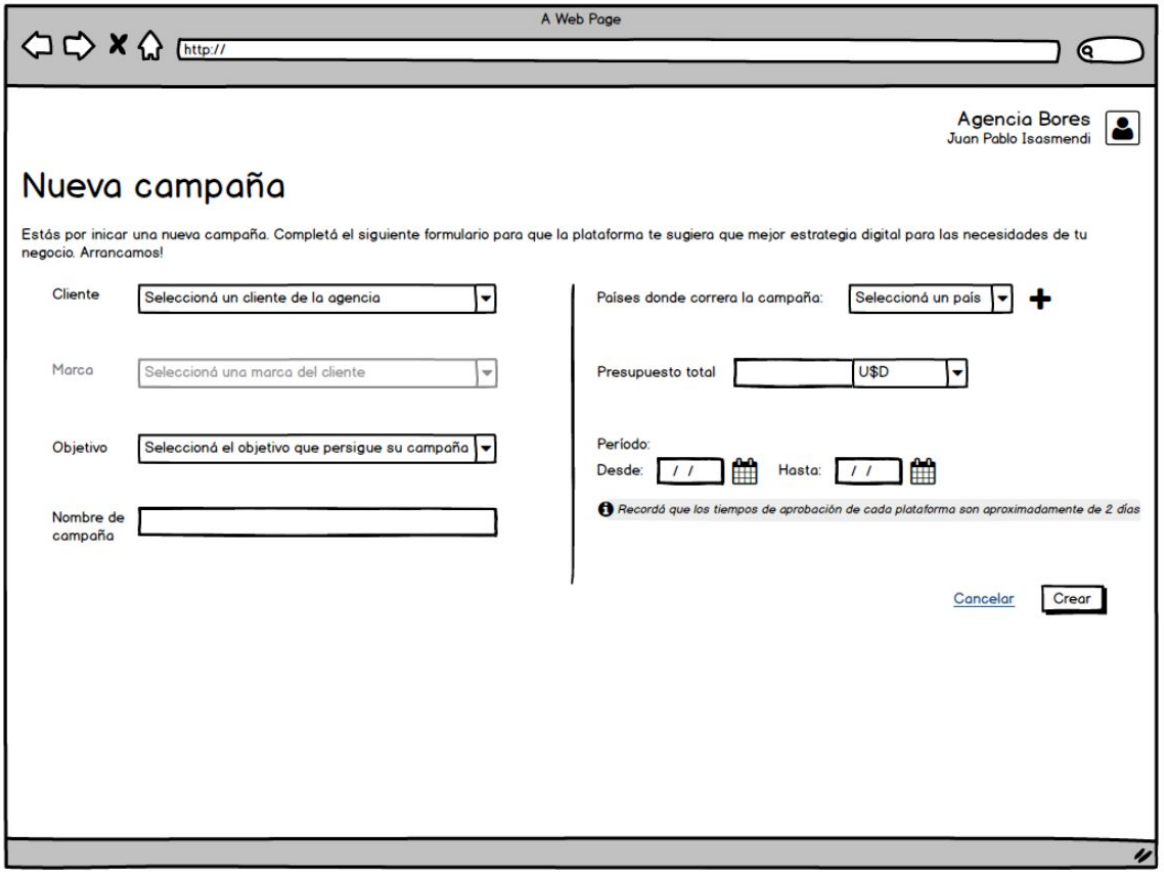

Fig. 42: Wireframe de pantalla de creación de campaña.

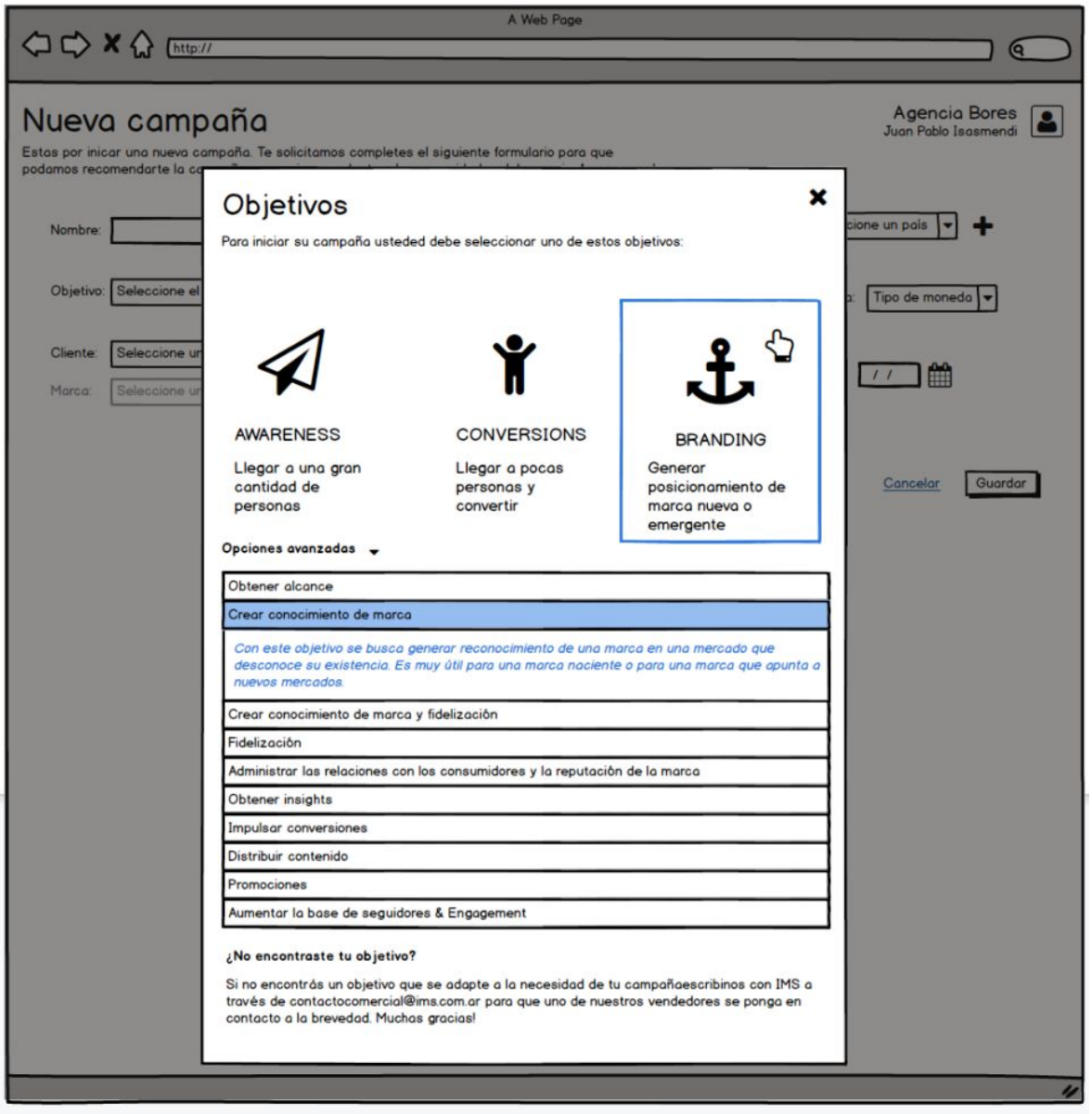

Fig. 43: Wireframe de selección de objetivos en la creación de campaña.

Fig. 44: Wireframe de recomendaciones de plataformas y productos basadas en los objetivos.

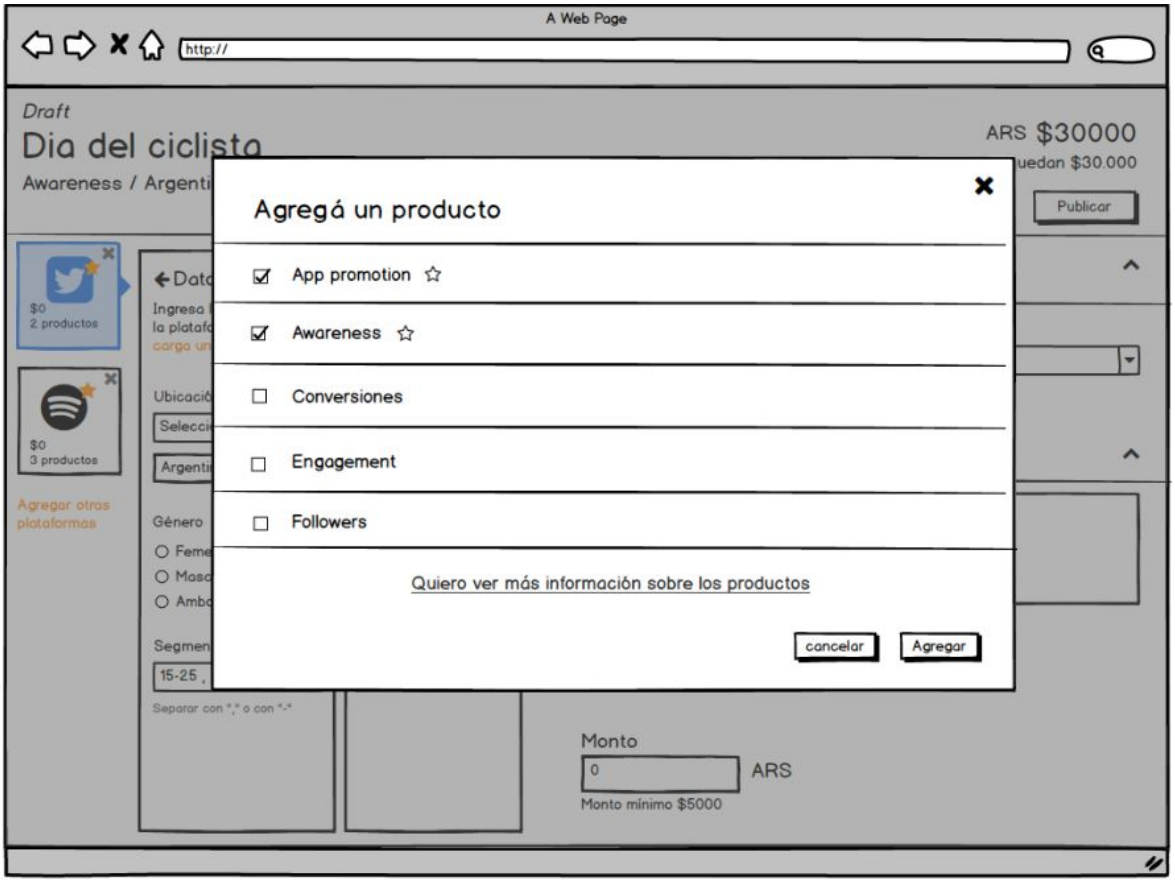

Fig. 45: Wireframe de selección productos de la campaña.

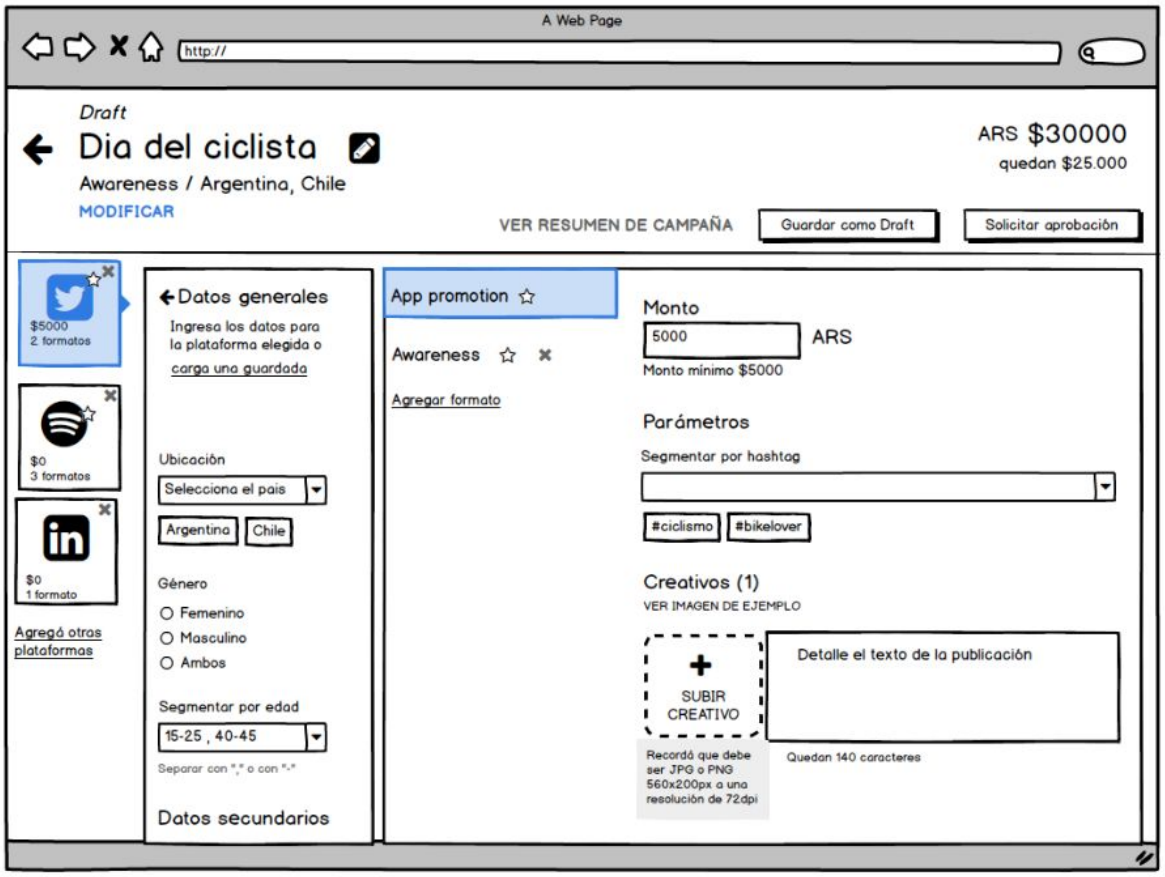

Fig. 46: Wireframe de la parametrización de la campaña.

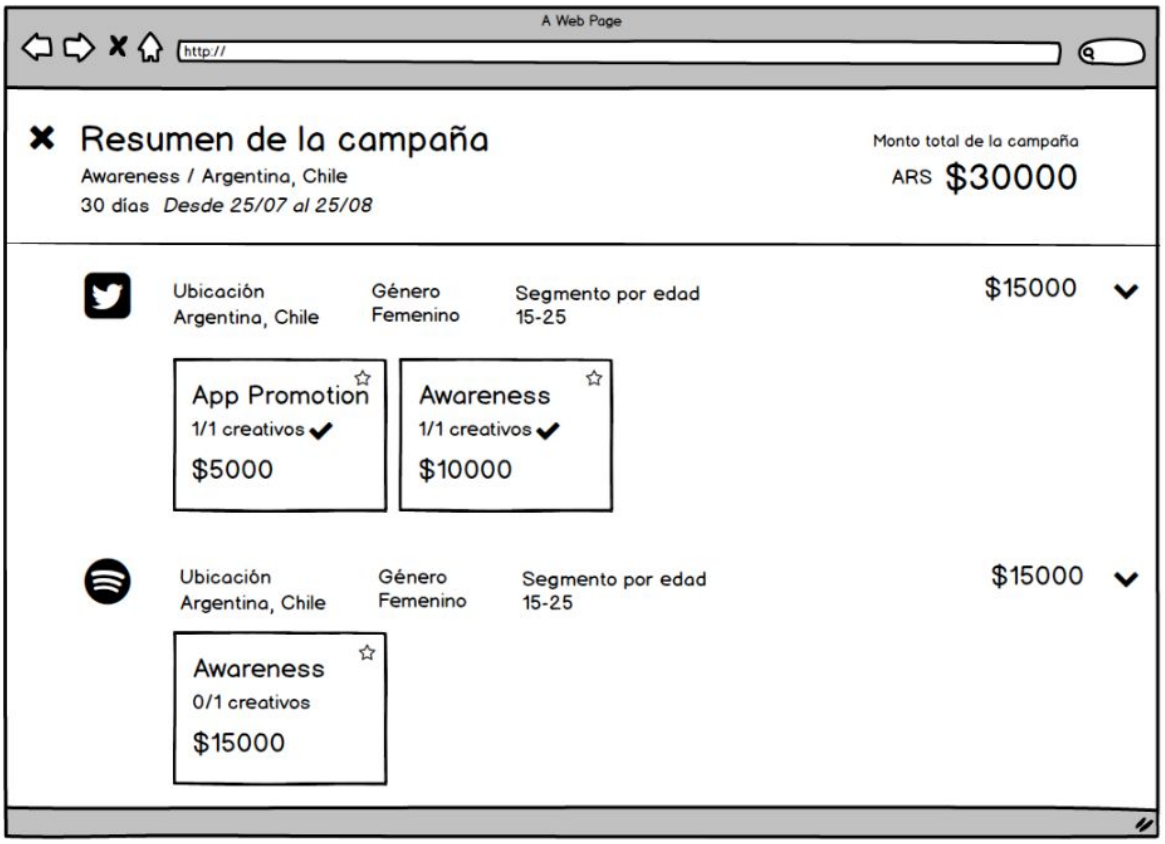

Fig. 47: Wireframe del resumen de campaña.

Finalmente, luego de los relevamientos, se trabajó sobre la personalidad de ONE y el branding[8] del mismo. De la mano de Material Design y de ilustraciones que reflejan la imagen empresarial, se logró un resultado felizmente aceptado por cada uno de los actores involucrados en el proceso de carga de campañas, ya que se generó una experiencia familiar para ellos.

[8] Branding: proceso mediante el cual se construye una marca, comprendiendo este como el desarrollo y mantenimiento de un conjunto de atributos y valores inherentes a la marca y por la que esta será identificada por su público

A modo de ejemplo, a continuación se mostrarán algunas de las pantallas del sistema ONE que resultaron a partir de los diseños anteriores y se expondrán las características principales del sistema que se desprenden de ellas.

La Fig. 48 muestra la pantalla del login, en la cual también se encuentra la opción de recuperar la contraseña.

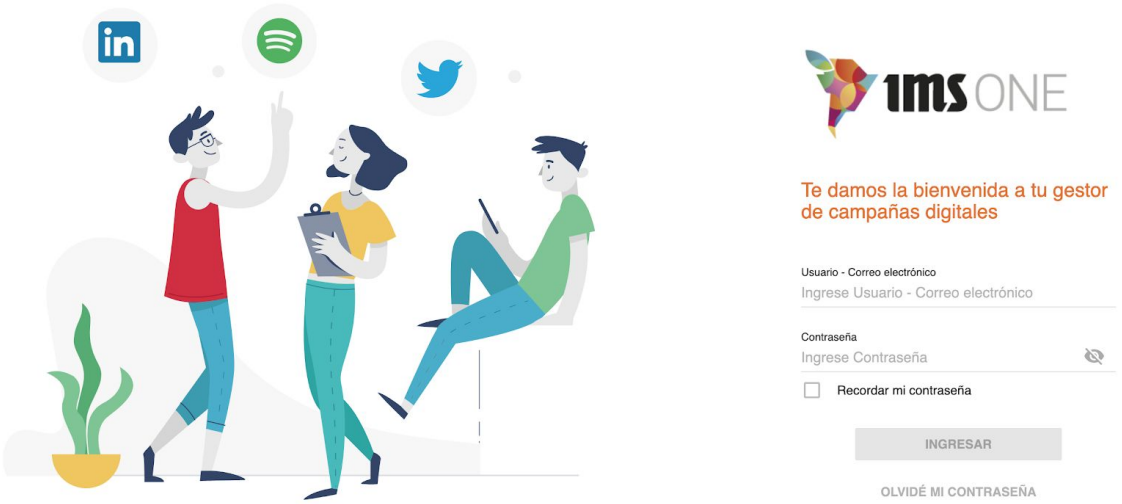

Fig. 48: Pantalla del login.

La Fig. 49 muestra el listado de campañas. Es la primer pantalla que ve el usuario luego de iniciar sesión. En el listado es posible filtrar las campañas por estados (Draft, Pendiente de publicación, Publicada o Finalizada), propietarios, plataformas y fechas de inicio y fin. Por defecto, el listado viene ordenado por fecha de creación pero también es posible ordenar por fecha de publicación.

Cada una de las campañas listadas muestran la siguiente información:

- Nombre.
- Anunciante al que pertenece.
- Presupuesto.
- Fechas de inicio y fin.
- Logos de las plataformas que posee configuradas.
- Una barra que refleja el avance de la campaña, basados en las fecha de inicio, fecha de fin y la fecha actual.

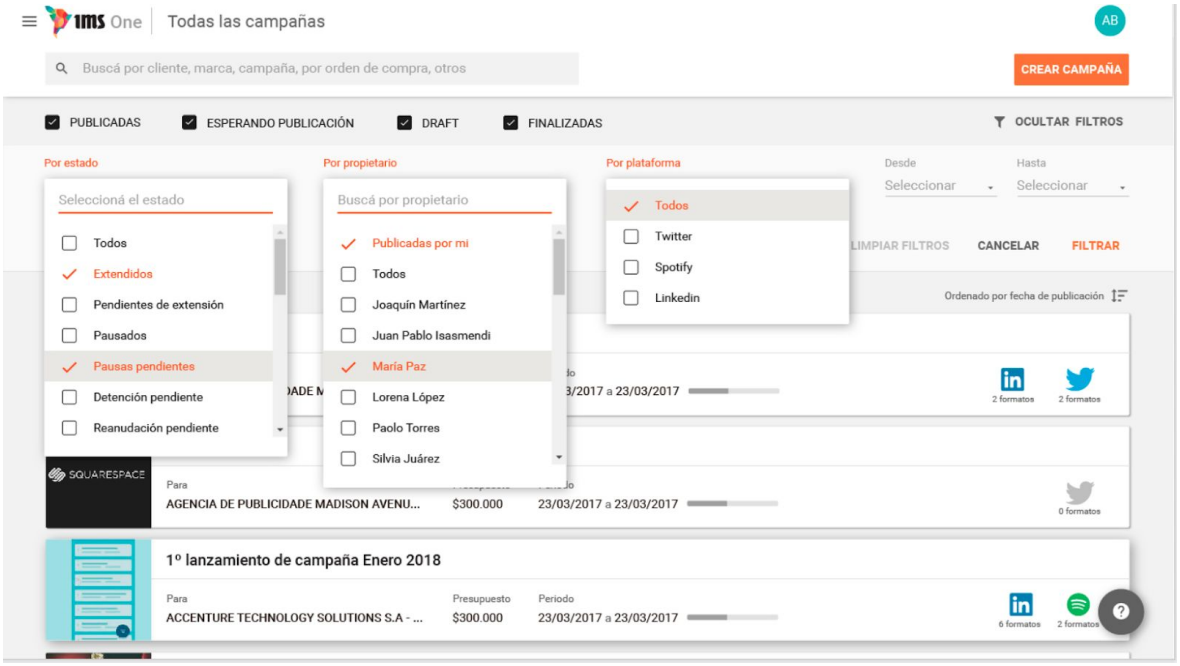

Fig. 49: Pantalla del listado de campañas.

La Fig. 50 muestra la creación de una campaña. Al momento de crear una campaña es necesario indicar la siguiente información:

- Nombre.
- Anunciante.
- Marca.
- Países donde correrá la campaña.
- Presupuesto.
- Moneda del presupuesto.
- Fecha de inicio.
- Fecha de fin.

Para finalizar la carga se pueden optar entre dos opciones:

A. Indicar directamente sobre que plataformas va a correr la campaña.
B. Indicar los objetivos de la campaña y utilizar las recomendaciones automáticas de plataformas y formatos del sistema. (Fig. 51)

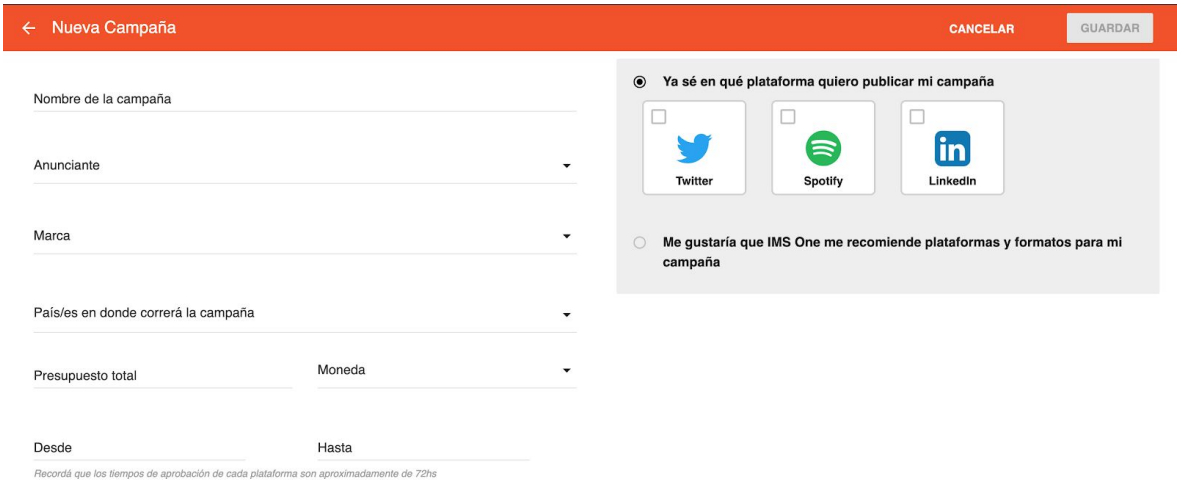

Fig. 50: Pantalla de creación de campaña.

Fig. 51: Pantalla de selección de objetivos.

En el caso de que en la creación de la campaña se opte por la selección de objetivos, el sistema mostrará un popup detallando cuales son las sugerencias del sistema basadas en los objetivos seleccionados (Fig. 52). En dicho popup el

usuario podrá aceptar total o parcialmente las recomendaciones de plataformas y productos así como también rechazarlas en su totalidad.

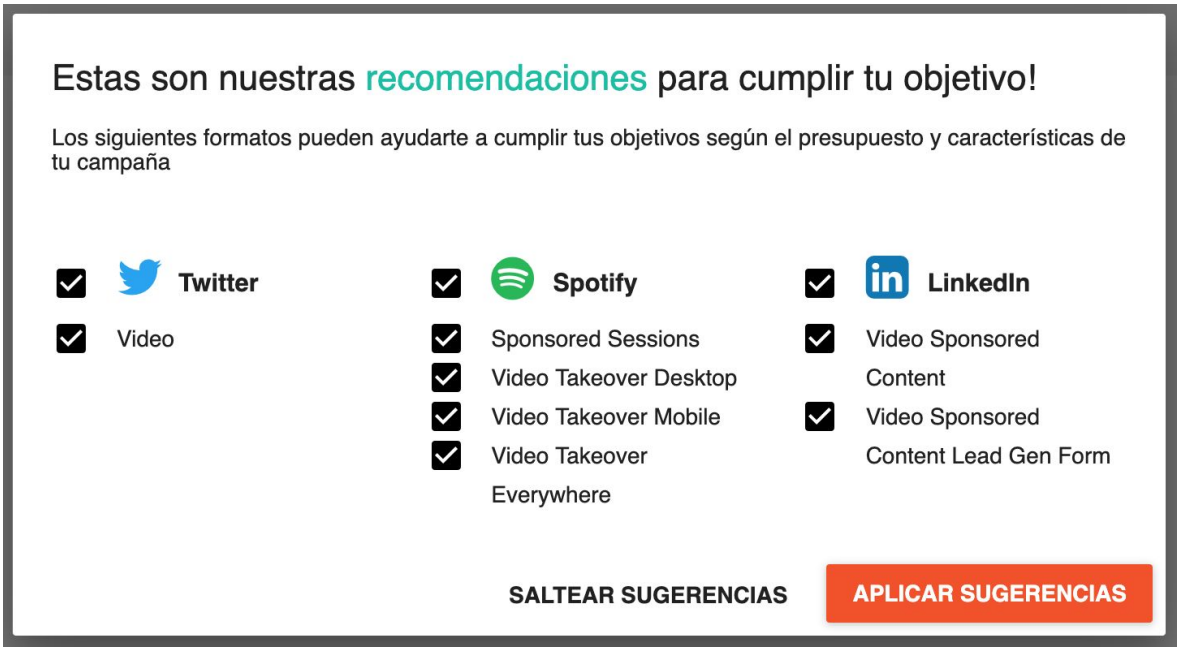

Fig. 52: Pantalla de recomendaciones de plataformas y productos basadas en los objetivos.

Una vez seleccionada una plataforma, aparecerán una serie de tarjetas que identifican las distintas tareas a realizar para poder publicar las campañas, indicando a su vez el estado de completitud de las mismas. Dichas tareas son:

- **Gestión de orden de compra:** Aquí es donde se carga la orden de compra con la cual la agencia/anunciante contrata lo servicios.
- **Segmentación:** Aquí es donde se indica la audiencia a la cual se va a apuntar la campaña.
- **Formatos y creativos:** Aquí es donde se cargan los formatos y creativos de la plataforma.

Existe un caso particular para Twitter (Fig. 53), en el que antes de poder empezar a cargar los datos de las campañas es necesario relacionar la campaña con un usuario de Twitter y de esta forma poder consumir información para la creación de la campaña como puede ser segmentaciones o utilizar tweets ya creados.

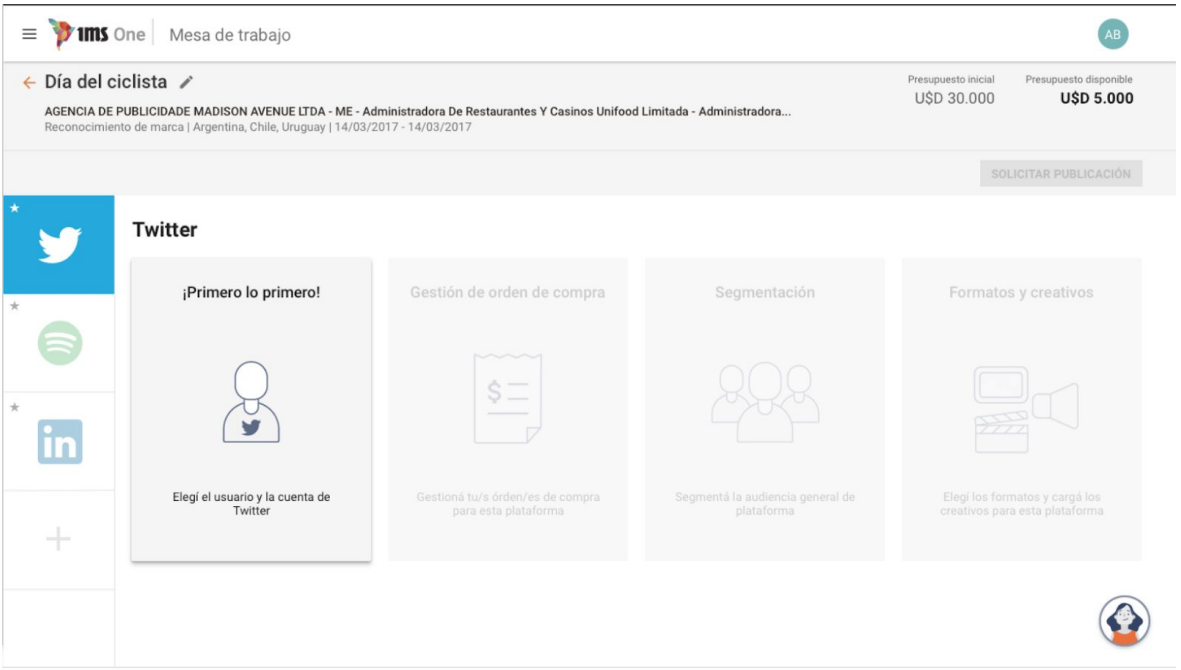

Fig. 53: Pantalla de tareas para el alta de campaña.

En cuanto a la segmentación de la campaña existen 3 formas de hacerlo (Fig. 54):

- Utilizar una audiencia predefinida por el sistema.
- Utilizar una audiencia previamente creada y guardada por el usuario autenticado.
- Crear una audiencia nueva libremente.

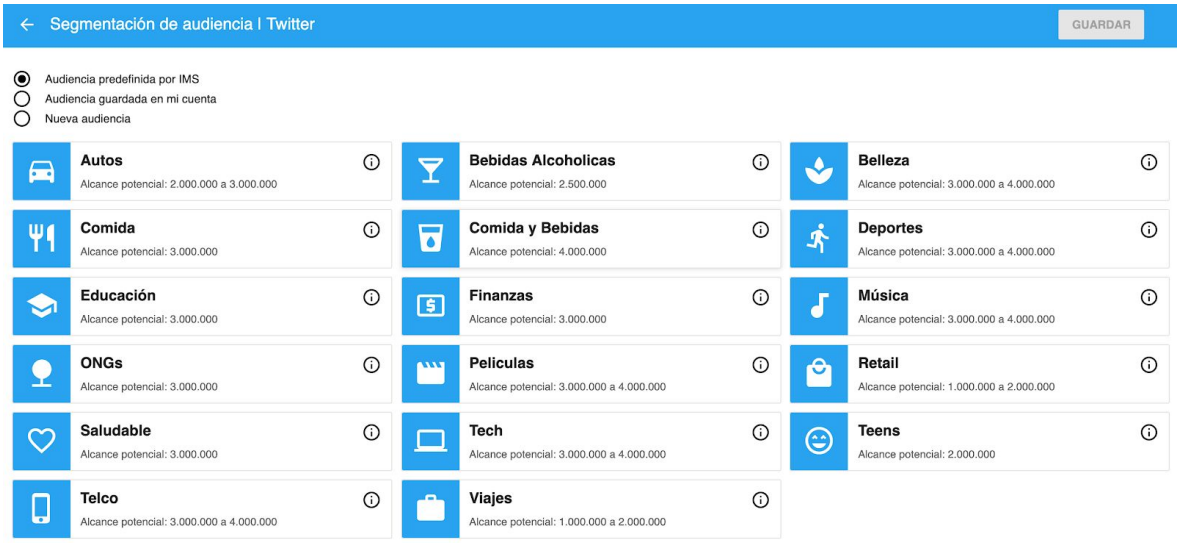

Fig. 54: Pantalla de segmentaciones predefinidas

Cada plataforma tiene sus propias segmentaciones, lo que permitirá apuntar la campaña a un público específico. La Fig. 55 muestra un ejemplo de creación de segmentación nueva para Twitter.

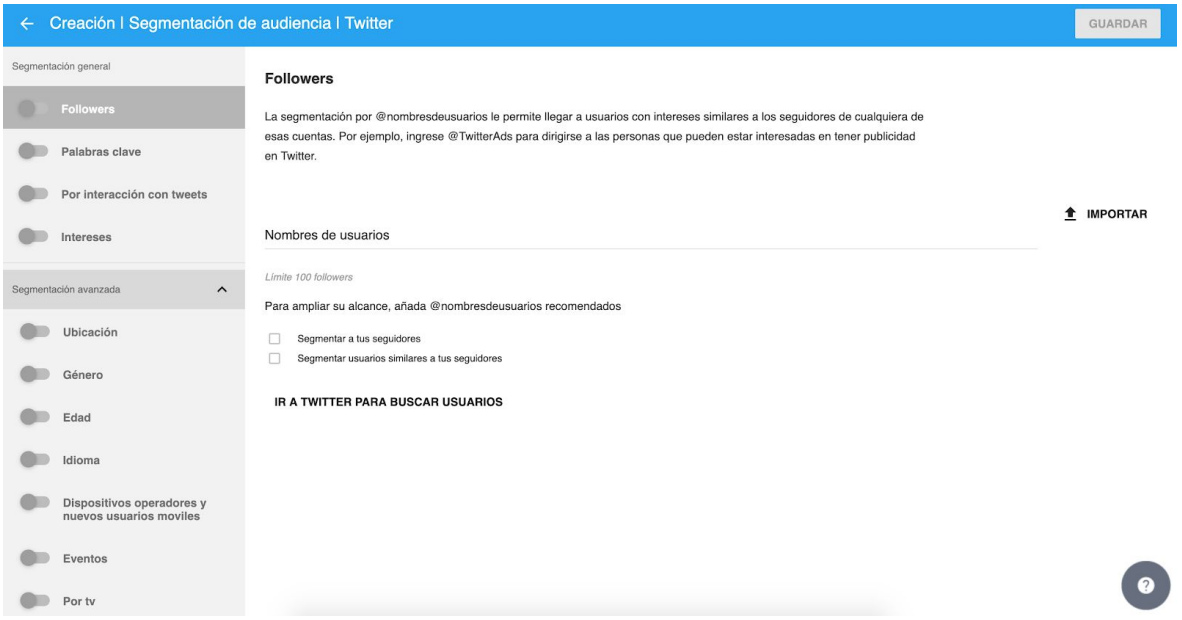

Fig. 55: Pantalla de edición de segmentación.

Cada una de las plataformas dispone de distintos formatos en los cuales se pueden lanzar las campañas digitales. En general, para cada formato, independientemente de la plataforma, es necesario indicar el presupuesto que se

le va a imputar, el presupuesto diario y la forma en que va a ser consumido. En algunos casos, como Spotify, esta información debe ser definida por cada uno de los países en los cuales se lanzará la campaña. La Fig. 56 muestra un ejemplo de la información necesaria por producto para Twitter.

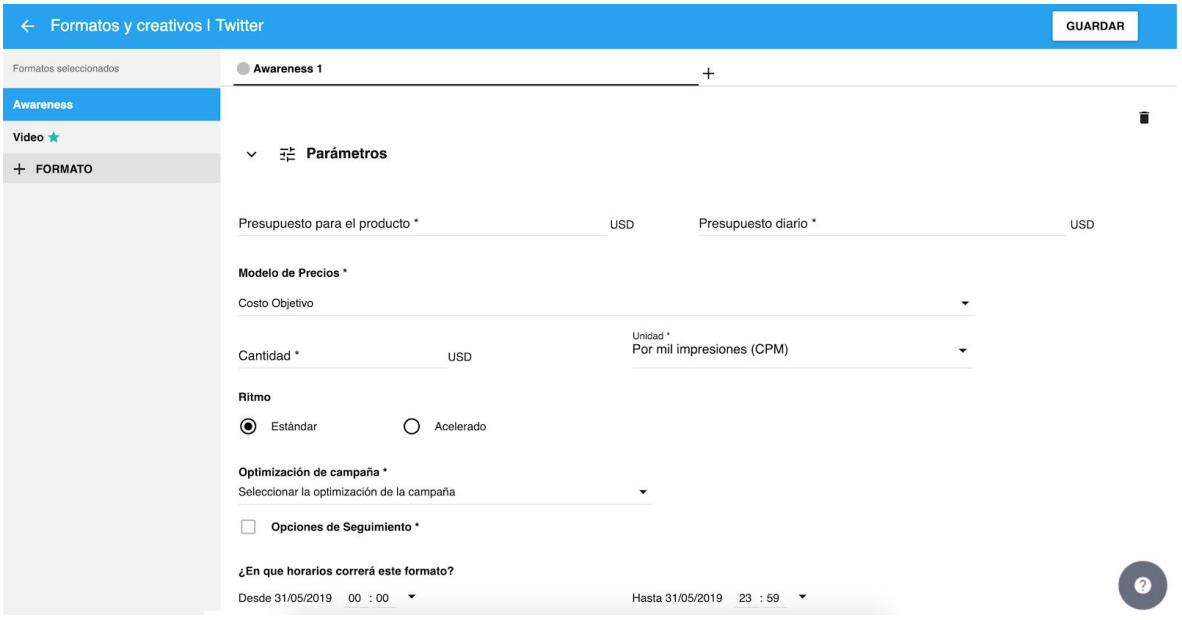

Fig. 56: Pantalla de parametrización del formato.

Finalmente está la carga del creativo, que es lo que terminarán viendo los usuarios de la plataforma como contenido promocionado. El creativo generalmente está compuesto por texto y algún elemento multimedia (imagen, audio o video). Cada plataforma define ciertos tipos de creativos según el tipo de formato elegido. La Fig. 57 muestra un ejemplo de carga de creativo para un formato de Twitter.

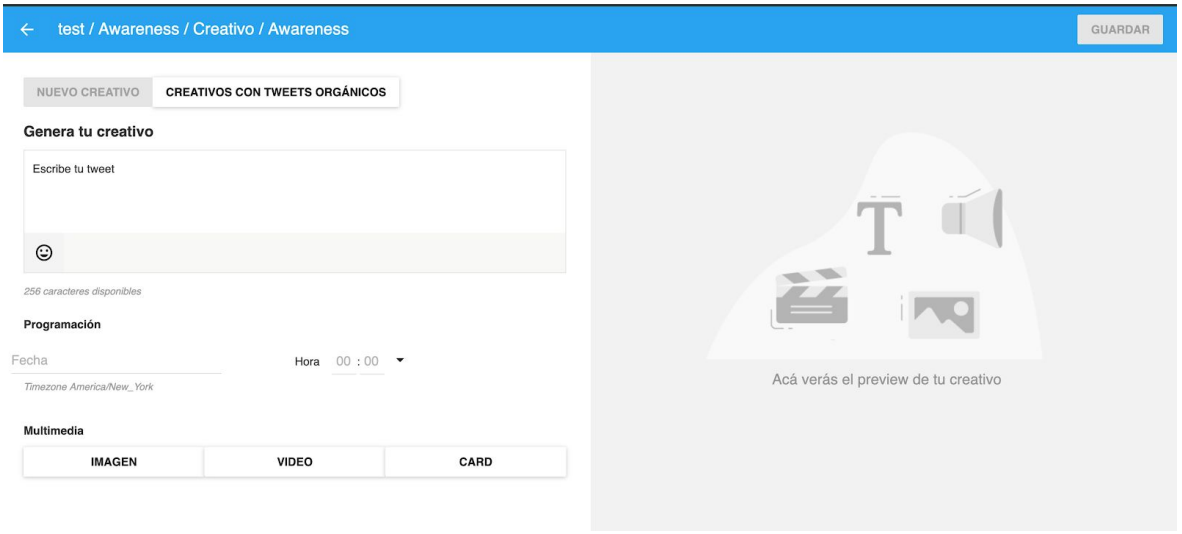

Fig. 57: Pantalla de alta de creativo.

A medida que se van completando las distintas tareas, éstas van cambiando de estado reflejando su estado. La Fig. 58 muestra el estado de las tareas, indicando cuales están completas y cuáles no.

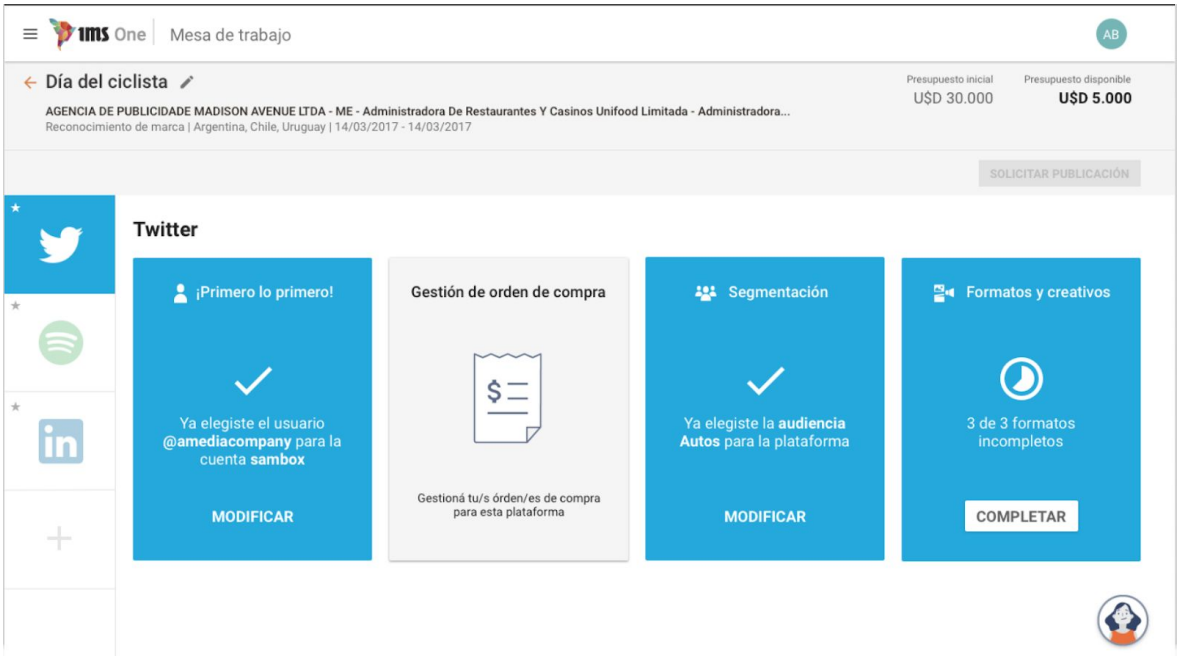

Fig. 58: Pantalla de resumen de las tareas para el alta de campaña.

CAPÍTULO 6. Conclusiones y líneas de trabajo futuras

6.1 Conclusiones

A lo largo de este informe de tesina se ha analizado el impacto que tuvo la Web 2.0 en el marketing tradicional, haciendo surgir el concepto de marketing digital. Se analizaron las distintas herramientas que posee el marketing digital hoy en día y se ha hecho principal hincapié en el rol fundamental que poseen las redes sociales para el lanzamiento de campañas publicitarias.

Como parte de este trabajo, se analizaron distintas tecnologías de vanguardia para el desarrollo de software, como Node.js y React, así como también conceptos técnicos fundamentales de JavaScript y el sistema REST.

Para cada una de estas etapas, se ha descripto oportunamente un marco teórico, y se han expuesto los desafíos encontrados y su forma de resolución.

El desarrollo de ONE implicó tres grandes desafíos. Por un lado el desafío técnico de desarrollar un sistema íntegramente en JavaScript, con tecnologías sumamente jóvenes como el caso de React. Por otro lado el desafío de realizar un sistema fácilmente adaptable dado que el core del negocio, las redes sociales, continúan evolucionando y desarrollando nuevos productos día a día. El tercer y más importante desafío fue la interfaz y experiencia de usuario ya que se debía lograr una experiencia igual o superior a las que ofrecían las distintas plataformas actualmente.

En conclusión, hoy en día ONE es una de las principales herramientas que posee IMS para poder desarrollar su negocio. Gracias al trabajo de UX realizado hoy, ONE está posicionado como una marca y es el eje central de la empresa.

6.2 Líneas de trabajo futuras

Al día de hoy ONE permite realizar campañas para toda América Latina en Spotify, Twitter y LinkedIn, ofreciendo la totalidad de sus productos y creativos. Las principales líneas de trabajo futuras incluyen:

- Integrar nuevas plataformas como Snapchat, Twitch o Foursquare.

- Expandir el mercado incluyendo nuevos países en los cuales se puedan trabajar, sobre todo en América del Norte.
- Integrar con las APIs de todas las plataformas.
- Reforzar el trabajo con el equipo de UX para realizar más tests de usabilidad con los usuarios.
- Incorporar inteligencia de datos para predecir el comportamiento de una campaña, por ejemplo, estimar la llegada potencial que tendrá una campaña basada en la segmentación aplicada a la misma.

BIBLIOGRAFÍA

Alet I Vilaginés, J. (2001). *Marketing eficaz.com*. ISBN: 978-8480886192. Gestión 2000.

Amundsen, M. & Richardson, L. (2013). *RESTful Web APIs: Services for a Changing World*. ISBN: 978-1449358068. O'Reilly Media, Inc.

Armstrong, G. & Kotler, P. (2008). *Fundamentos de marketing.* ISBN: 978-9702611868. Pearson Education, Inc.

Banks, A. & Porcello, E. (2017). *Learning React: Functional Web Development with React and Redux*. ISBN: 978-1491954621. O'Reilly Media, Inc.

Bojinov, V. (2015). *RESTful Web API Design with Node.js*. ISBN: 978-1783985869. Packt Publishing Ltd.

Brown, E. (2016). *Learning JavaScript. JavaScript Essentials for Modern Application Development*. ISBN: 978-1491914915. O'Reilly Media, Inc.

Burgos, E., Cerezo, J., Cortés, M., Garolera, E. et al. (2009). *Del 1.0 al 2.0: Claves para entender el nuevo marketing*. ISBN: 978-8499160443. España: Bubok Publishing.

Calvo Fernandez, S. y Reinares Lara, P. (2001). *Estrategias de Márketing y comunicación interactivas.* Paraninfo Thomson Learning.

Cantelon, M., Harter, M., Holowaychuk, T. J. & Rajlich, N. (2013). *Node.js in Action.* ISBN: 978-1617290572. Manning Publications.

Celaya, J. (2011). *La empresa en la web 2.0*. ISBN: 978-8498751895. Gestión 2000.

Levine, F., Locke, C., Searls, D y Weinberger, D. (2000). *El manifiesto Cluetrain : el ocaso de la empresa convencional.* ISBN: 0-7382-0431-5. Ediciones Deusto, Perseus Books.

Nieto Churruca, A. y Rouhiainen, L. (2012). *La Web de Empresa 2.0*. ISBN: 978-8492570966. Global Marketing Strategies.

Powers, S. (2016). *Learning Node: Moving to the Server-Side*. ISBN: 978-1491943120. O'Reilly Media, Inc.

Redondo, M. y Rojas, P. (2017). *Cómo monitorizar las redes sociales.* ISBN: 978-8483569818. Editorial LID.

Richardson, L. & Ruby, S. (2007). *RESTful Web Services*. ISBN: 978-0596529260. O'Reilly Media, Inc.

Solís, A. (2016). *SEO. Las Claves Esenciales*. ISBN: 978-8441537286. Anaya Multimedia.

Stefanov, S. (2016). *React: Up & Running: Building Web Applications.* ISBN: 978-1491931820. O'Reilly Media, Inc.

Teixeira, P. (2013). *Professional Node.js*. ISBN: 978-1118185469. John Wiley & Sons, Inc.

Tilkov, S. & Vinoski, S. (2010). *Node.js: Using JavaScript to Build High-Performance Network Programs.* IEEE Internet Computing, Volume: 14, Issue: 6, Nov.-Dec. 2010. Recuperado de: https://ieeexplore.ieee.org/xpl/tocresult.jsp?isnumber=5617049.